UNIVERSIDAD COMPLUTENSE DE MADRID
FACULTAD DE INFORMÁTICA

Desarrollo de un sistema de predicción de riesgo en seguros de automóviles mediante técnicas de inteligencia artificial

Autores: Dámaso Sánchez Arenas y Eduardo Rodríguez De Castro Zaloña.

Tutores: José Ignacio Hidalgo Pérez y Óscar Garnica Alcázar.

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Agradecimientos

En primer lugar, agradecer a nuestros directores de proyecto José Ignacio y Óscar, por darnos la oportunidad de realizar el proyecto con ellos y poder ofrecernos conocimiento en un mundo que es tan complejo y esta tan en auge como es la inteligencia artificial. Por haber estado continuamente disponibles para posibles dudas y por estar siempre a nuestra disposición. Fuera de lo académico, agradecer el trato recibido el cual ha sido ejemplar y en muchas ocasiones nos ha servido como un gran impulso de motivación.

También agradecer a la universidad, por habernos facilitado todas sus instalaciones a lo largo de estos años y por habernos ayudado en todo lo que ha estado a su alcance.

Por otro lado, dar las gracias tanto a familiares como amigos que nos han estado ayudando durante un año complicado para nosotros y que han sido capaces de animarnos a que sigamos adelante a pesar de los momentos duros.

Resumen

Este trabajo de fin de grado consiste en el desarrollo de una herramienta para calcular mediante técnicas de inteligencia artificial el riesgo que tienen nuevos clientes de sufrir un accidente en sus vehículos. Principalmente, esta aplicación informática es una ayuda a las aseguradoras para intentar ajustar lo máximo posible los precios de los seguros. Las técnicas utilizadas en este proyecto han sido las redes neuronales y el Random forest. Todo el proyecto ha sido implementado en Python, tanto la parte de la interfaz gráfica, con la que conseguimos que el software tenga una visión sencilla y amena, como la parte donde se realizan predicciones. El software puede ser usado por dos distintos perfiles. Un perfil técnico para personas expertas en esta área en el que pueden crear su propio modelo de predicción usando las características (Features) que crean más oportunas y un perfil funcional para usuarios que no tienen tanto conocimiento acerca de estas tecnologías. Podrán cargar un modelo ya creado y predecir en base a él. En este proyecto, se ha conseguido tener un producto con una interfaz que cumple perfectamente con las funcionalidades del software y en la parte del funcionamiento, tiene la capacidad de predecir riesgos gracias a las técnicas de inteligencia artificial mencionadas anteriormente.

Palabras clave: Software, inteligencia Artificial, riesgo, redes neuronales, Random forest, Python, predicciones.

Abstract

This paper is aimed at the development of a tool that, through Artificial Intelligence, calculates new clients' risk of suffering a vehicle accident. The objective of this software application is to support insurance companies in adjusting prices. For this project, neuronal networks and Random Forest techniques have been used. The whole project has been implemented in Python, the graphic interface, which allows a pleasant and simple usage of the application, and the predictor. Software is developed with two different user profiles in mind; a technical profile, for experts in this area who wish to create their own prediction model using those Features they consider the most appropriate, and a functional profile for users who don't have a great knowledge on these technologies which will allow them to upload an already existent model and obtain predictions based on it. In this project, a product with an interface that perfectly fulfils the software functionalities has been achieved. Moreover, within the operational part of the project, the software has the capacity of predicting risks through Artificial Intelligence techniques.

Keywords: Software, Artificial Intelligence, risk, neuronal networks, Random forest, Python, predictions.

Índice general

Capítulo 1

Introducción

1.1. Descripción del problema

Hoy en día, en el mundo de las aseguradoras, el precio de los seguros está basado en el análisis de los actuarios, lo cual en muchas ocasiones es muy costoso. Aunque parezca irreal, pese a todas las tecnologías que tenemos, no se utilizan técnicas informáticas avanzadas para ayudar en este análisis.

Para hacernos una idea, actualmente el precio de un seguro se calcula en base a la estadística de dos variables vitales: la probabilidad de que ocurra un imprevisto y el coste medio que supondrá solucionarlo. [1]

El encargado de realizar estos cálculos es el denominado "Actuario". Como sabemos, no es fácil calcular la probabilidad de que ocurra un imprevisto y es por ello que muchas veces los seguros se basan en un perfil de cliente muy genérico en vez de individualizar con el cliente concreto que van a tratar.

Por otro lado, como bien sabemos, hoy en día, está totalmente en auge las técnicas relacionadas con la inteligencia artificial como el machine learning, redes neuronales, etc. Esto es debido a que se están consiguiendo resultados muy positivos en diversas problemáticas.

Pese a que muchas de las técnicas utilizadas llevan existiendo mucho tiempo, es ahora, cuando se están aplicando en problemas reales. Esto es así debido al gran aumento en la capacidad de cómputo de las últimas décadas y a la facilidad que existe a día de hoy en la escalabilidad de los datos gracias tanto a la nube como al big data. Es fundamental tener en cuenta que la inteligencia artificial debe estar totalmente ligada a los datos y viceversa.

Viendo esta carencia, nuestro objetivo es facilitar el análisis de los actuarios con una aplicación informática basada en técnicas de inteligencia artificial y que así puedan adaptarse de una manera mucho más específica al cliente.

1.2. Idea del proyecto

En este proyecto hemos realizado una aplicación informática con el cual vamos ayudar a las compañías de seguro a predecir de manera más óptima y con muchas más garantías el riesgo de los clientes para que de esta manera puedan estipular los precios de sus tarifas y así sean más ajustadas e individualizadas para cada uno de los clientes.

Para ello, nos hemos basado en distintas técnicas de inteligencia artificial. Los datos utilizados para que nuestros clasificadores funcionen correctamente han sido facilitados por aseguradoras reales en formato *"Excell"* y donde gracias a ellos, hemos conseguido una muestra suficiente para obtener un clasificador.

Toda la aplicación está implementada en Python3 y desarrollada en el entorno de jupyter notebook.

La predicción estará fundamentada en varios modelos de clasificación:

1. Redes neuronales
2. Random forest

Las aseguradoras tendrán cierta libertad a la hora de configurar el programa. De esta manera, podrán dar prioridad a ciertas características que para ellos sean más importantes. Las aseguradoras podrán configurar la aplicación fácilmente gracias a nuestra interfaz, sin necesidad de entender lo que hace el programa en su interior.

1.3. Conclusión

Como hemos comentado, a las aseguradoras no les resulta nada sencillo calcular las tarifas para sus clientes, esto es debido a que las predicciones del riesgo de sus clientes están basadas en el trabajo de una persona en vez de en una aplicación informática.

Creemos que es una carencia no hacer uso de ninguna de la gran variedad de tecnologías que hay disponibles hoy en día.Por eso mismo, hemos visto una oportunidad para implementar una herramienta innovadora, para poder aprender sobre nuevos lenguajes de programación que no habíamos visto previamente y para aprender sobre un apasionante mundo, la inteligencia artificial.

Con nuestro producto intentaremos ayudar a las aseguradoras facilitando una aplicación que calcula distintas predicciones del riesgo de un cliente y de esta manera, facilitar el trabajo a los actuarios que podrán basarse en ello para establecer un precio.

(Versión inglesa de la introducción en el capítulo 8)

Capítulo 2

Planificación del trabajo

Con el fin de conseguir los objetivos, la planificación seguida fue la siguiente:

1. Estudio individual por parte de los miembros del equipo con el fin de tener un conocimiento global acerca del machine learning y las técnicas utilizadas en este trabajo.
2. Puesta en común por parte de los miembros del equipo y toma de decisiones acerca de las tecnologías a usar junto con los directores del proyecto.
3. Instalación y despliegue de la herramienta: Anaconda – Jupyter notebook. (Plataforma de trabajo).
4. Análisis del trabajo global a realizar y división del mismo para poder ir abarcándolo pasó a paso.
5. Diseño de los bocetos iniciales realizados por *balsamiq* para tener una visión inicial de la interfaz.
6. Estudio acerca de Python ya que es el lenguaje principal del programa. Profundizando en las técnicas de inteligencia artificial que debemos utilizar para nuestra aplicación.
7. Estudio de las características a usar en el modelo de datos para comenzar la limpieza de datos.
8. Programación de manera individual con dos subobjetivos principales:
    a. Interfaz gráfica (Front-end)
    b. Limpieza y tratamiento de datos (Back-end)
9. Comienzo de creación de una red neuronal básica con el fin de probar el tratamiento de los datos.
10. Unión de los subobjetivos para conseguir tener una primera aplicación funcional con la que ya se pueda comenzar el estudio de los resultados obtenidos.
11. Realización de pruebas con la matriz de confusión de la red neuronal para evaluar el rendimiento.
12. Mejora acerca de la interfaz para que sea lo más completa y amena posible.
13. Estudio profundo acerca del clasificador Random forest.
14. Implementación del resto de clasificadores (Random forest).
15. Pruebas del Random forest y corrección de errores.
16. Integración del algoritmo Random forest en la aplicación final.
17. Pruebas de funcionamiento de la aplicación final y corrección de posibles fallos.

Cabe destacar que la realización de la memoria se fue haciendo de forma paralela al desarrollo descrito anteriormente.

Comentar también, que alguna de estas tareas se realizaron de manera común por el equipo, y que otras, fueron abordadas por un único miembro, según lo acordado en la planificación.

(Versión inglesa en capítulo 9)

Capítulo 3

Tecnologías utilizadas

A lo largo de este capítulo describiremos cuáles han sido las tecnologías usadas en este proyecto. Argumentaremos el motivo de la elección de dichas tecnologías y explicaremos el proceso de instalación de las mismas.

3.1. Entorno Windows

El Sistema operativo donde hemos implementado nuestro producto es Windows 10. Cabe destacar que la aplicación debería de poder funcionar en cualquier otra versión de Windows.

La elección fue debido a que Windows es el sistema operativo con el que trabajamos día a día, y ya que el sistema operativo en este proyecto no es una pieza relevante, consideramos que era oportuno trabajar con un entorno que nos fuese cómodo.

3.2. Python

Python es el lenguaje de programación por excelencia para este tipo de proyectos debido a que existen multitud de bibliotecas que facilitan mucho el trabajo. Su principal virtud es la legibilidad de su sintaxis ya que es muy expresiva e intuitiva, es por ello, que muchas personas irónicamente dicen la frase de "Si sabes inglés, sabes Python".

La elección de un lenguaje de programación para este proyecto fue una tarea sencilla ya que Python es el lenguaje por excelencia para el machine learning y todo este tipo de herramientas de inteligencia artificial.

A continuación, comentamos el proceso de instalación:

- En primer lugar, tuvimos que descargarnos desde la página oficial de Python (https://www.python.org/), la última versión que haya disponible. En nuestro caso, la versión 3.7.1.
- A continuación, se debe abrir el archivo .msi descargado y se debe seguir todos los pasos de la guía.
- Por último, una vez finalizada la instalación, para cerciorarnos que lo hemos realizado correctamente, debemos ejecutar en el terminal de Windows el comando "python –version". Si todo ha ido correctamente, en el terminal debe salir la versión que hayamos instalado como podemos ver en la figura 3.2

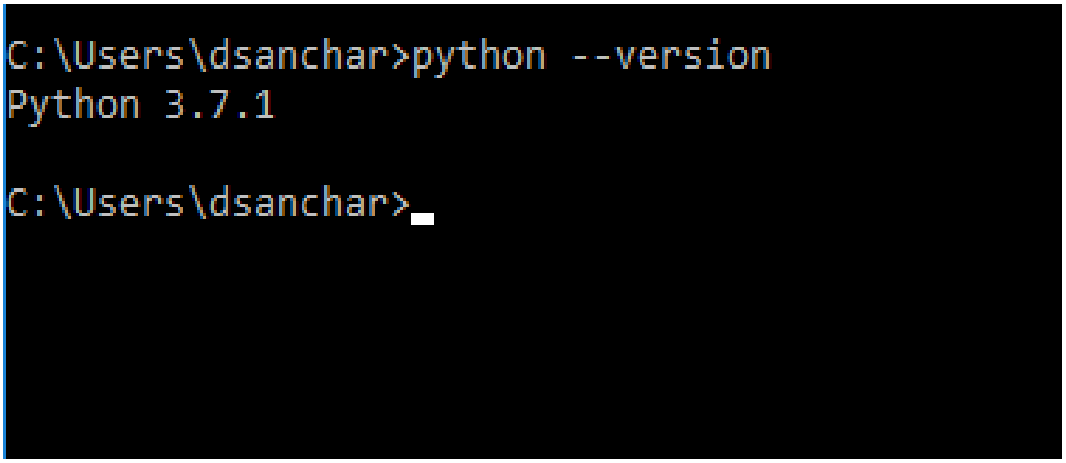


Figura 3.1: Versión de Python instalada.

En nuestro caso, queríamos realizar una aplicación en un entorno actualizado y que nos fuese útil para nuestro futuro laboral. Por ello, decidimos usar el entorno jupyter notebook.

Jupyter notebook es un entorno el cual no necesita ser descargado, sino que es un entorno interactivo web para la ejecución de código, lo cual está muy de moda en el mundo profesional en estos momentos. Además, otra de las razones por las que usamos este entorno es que se pueden incluir, análisis de datos, explicación de datos, etc con mucha facilidad. En la figura 3.2 podemos ver una muestra de como es jupyter notebook.

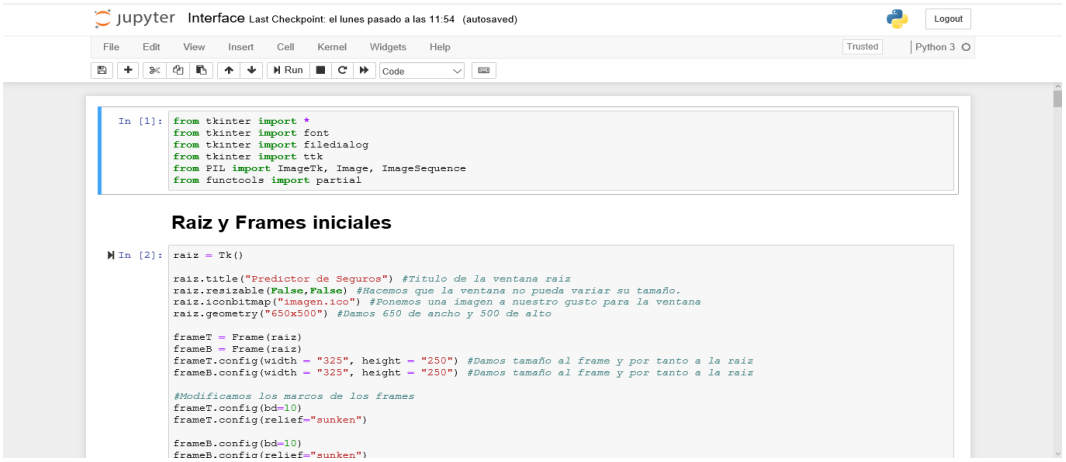


Figura 3.2: Interfaz jupyter notebook.

Este entorno no es algo que se pueda desplegar trivialmente, sino que se necesita de un software que lo distribuya. Para ello, usamos Anaconda.

Anaconda, es una distribución libre y abierta de los lenguajes Python y R. Se utiliza frecuentemente para la ciencia de datos y el aprendizaje automático, es decir, justo lo que necesitamos para realizar nuestro proyecto.

A continuación, dejamos una imagen en la figura 3.3 de cómo sería el proceso para entrar en jupyter notebook desde anaconda.

Figura 3.3: Acceso a Jupyter Notebook.

Una vez puesto el comando *"jupyter notebook"* en anaconda, automáticamente se nos abrirá una página web con el entorno interactivo de jupyter notebook como podemos ver en la figura 3.4.

Una vez aquí, ya podemos comenzar a navegar por jupyter notebook.

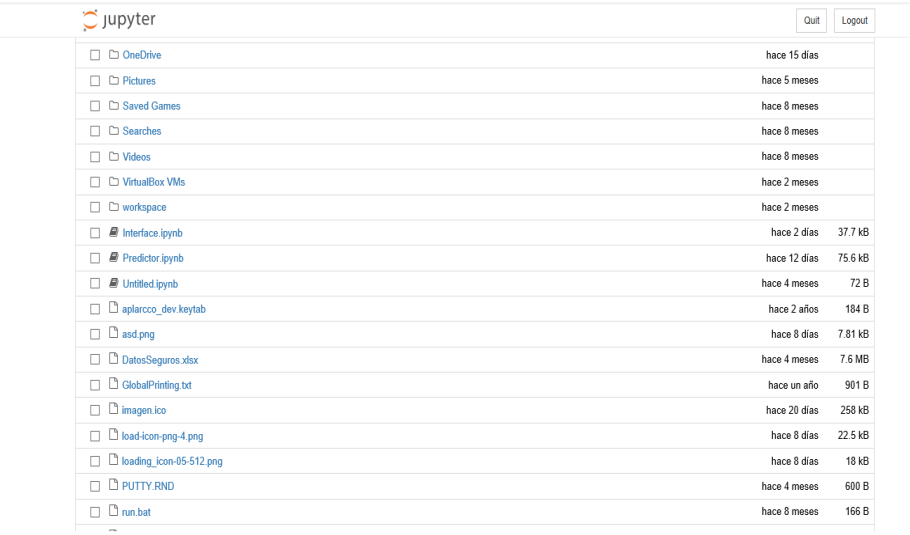


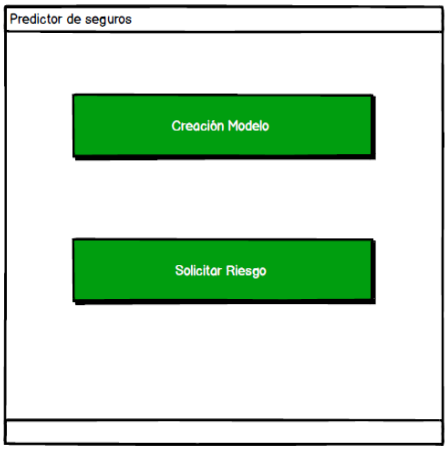
Figura 3.4: Visión jupyter notebook.

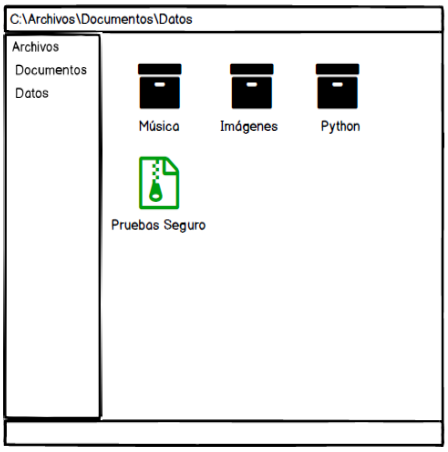
Capítulo 4

Front-end de la aplicación

Para la realización de la interfaz, lo primero que realizamos fue la creación de unos *mockUps* en la página balsamiq [2]. De esta manera, pudimos obtener una primera visión de la interfaz y conseguimos ver un primer boceto de como iba a ser nuestra interfaz.

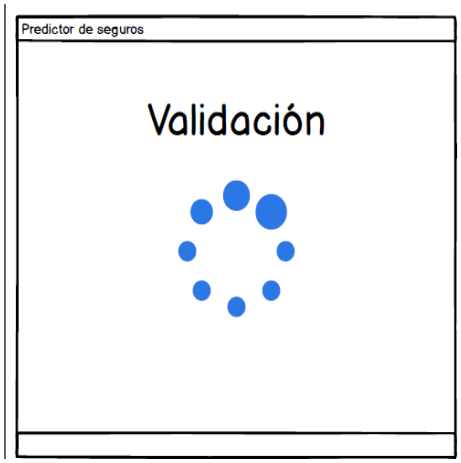
Estos fueron los primeros bocetos que hicimos para la realización de la interfaz:




C:\Archivos\Documentos\Datos
Archivos
Documentos
Datos
Música
Imágenes
Python
Pruebas Seguro

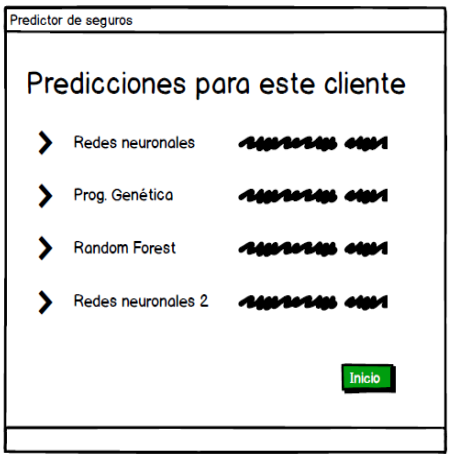

Predictor de seguros
Preprocesando


Predictor de seguros
Tratamiento de los datos


Predictor de seguros
Validación


Predictor de seguros
Modelo creado con éxito


Predictor de seguros
Formulario Cliente
Nombre
Jacinto
Apellidos
Lopez
e-mail
jalopez@gmail.com
Edad
57
Años carnet
34
Aceptar



Una vez que ya teníamos clara una primera idea de como queríamos que quedara la interfaz, empezamos a implementarla en el lenguaje Python. Para ello, usamos la biblioteca *tkinter* la cual nos ha dado mucha facilidad a la hora de crear numerosos *widgets* debido a la amplia información que hay acerca de ella en Internet.

En la siguiente sección, vamos a enseñar nuestra interfaz real e iremos explicando como sería una correcta utilización de nuestra aplicación.

4.1. Funcionalidad

Nuestro aplicación tiene 3 funciones principales:

1. Creación del modelo: Funcionalidad pensada para crear un modelo con las características que el usuario crea oportuno. Una vez creado, el usuario podrá guardar aquellos modelos que cumplan sus necesidades para posteriormente usarlos.

2. Carga del modelo: Funcionalidad pensada para cargar un modelo ya creado y guardado. El último modelo cargado será el que utilicemos a la hora de predecir el riesgo de un nuevo cliente.

3. Solicitar Riesgo: Funcionalidad pensada para calcular el riesgo de un nuevo cliente. Esta funcionalidad se basará en el último modelo cargado por parte del usuario. En caso de que no se haya cargado ningún modelo con anterioridad no se podrá solicitar riesgo para un nuevo cliente.

A continuación, en la figura 4.1 podemos ver nuestra ventana de inicio. En ella, se pueden ver tres botones los cuales cada uno de ellos inicia cada una de las funcionalidades mencionadas anteriormente.

En este caso, la funcionalidad de solicitar riesgo esta deshabilitada ya que no existe ningún modelo cargado y por tanto no se puede solicitar el riesgo de un nuevo cliente.

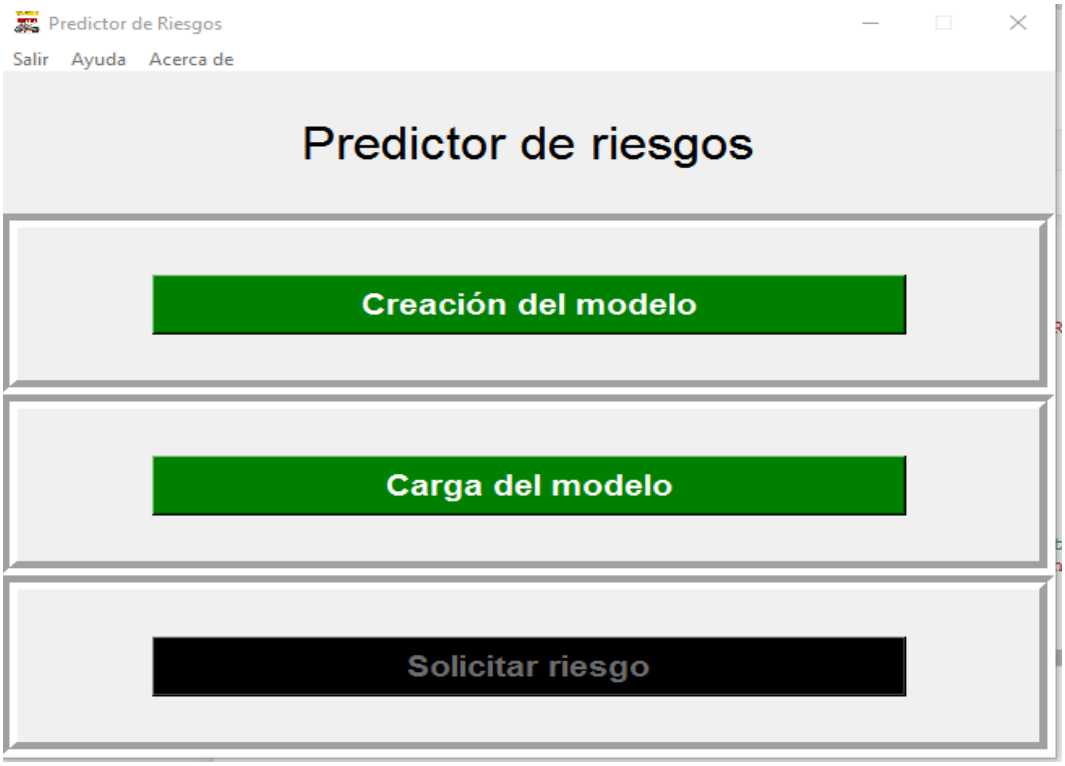


Figura 4.1: Ventana de inicio.

En primer lugar, vamos a explicar el menú que tenemos desarrollado y el cual estará operativo continuamente en la parte superior de todas las ventanas.

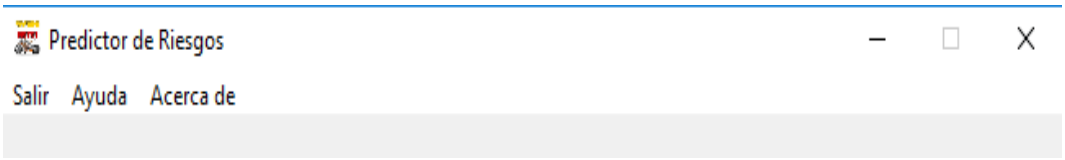


Figura 4.2: Menú de la aplicación.

Como podemos ver en la figura 4.2, nuestro menú consta de tres partes, las cuales mostramos a continuación.

- Salir → Ventana emergente que nos pregunta si deseamos salir. En caso afirmativo, estarás poniendo punto y final a la ejecución de la aplicación.

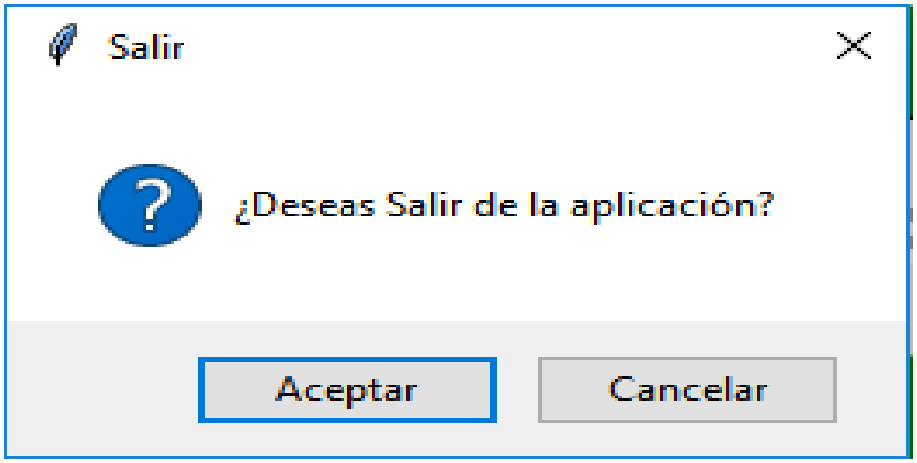


Figura 4.3: Ventana Emergente: Salir.

- Ayuda → Ventana emergente que informa que existe una memoria acerca de la aplicación donde se puede consultar cualquier duda.

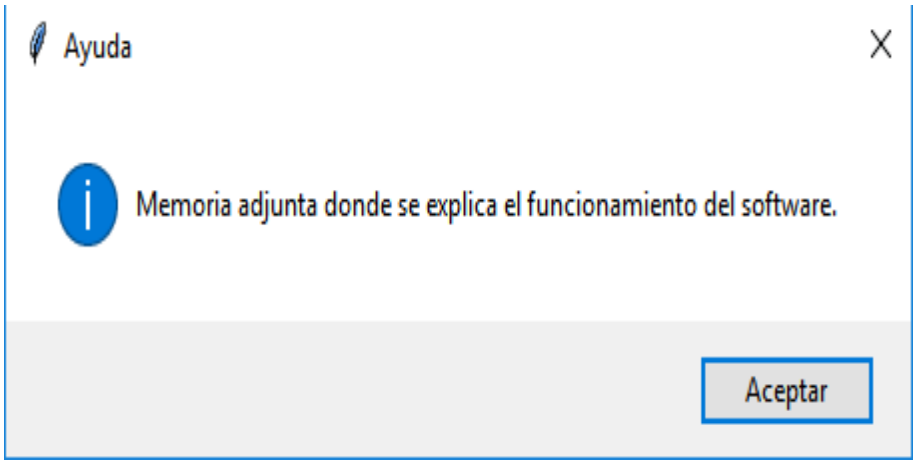


Figura 4.4: Ventana Emergente: Ayuda.

- Acerca de → Ventana emergente que informa sobre los creadores de la aplicación.

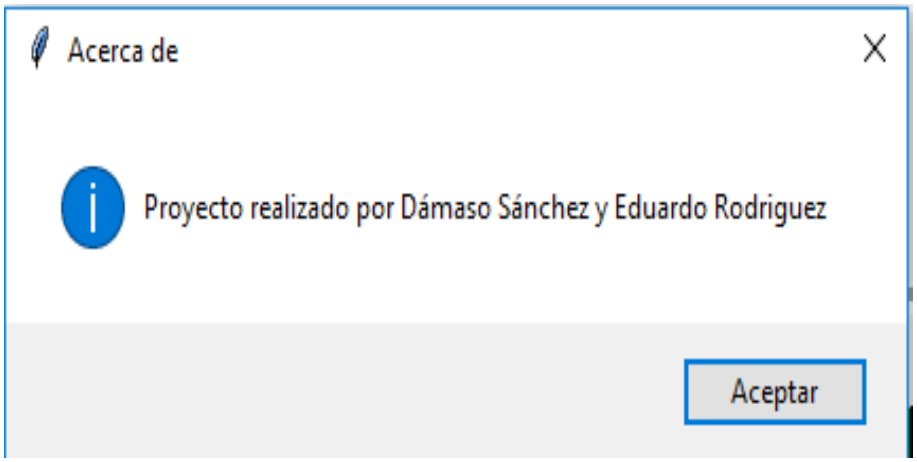


Figura 4.5: Ventana Emergente: Acerca de.

Una vez vista la ventana inicial y el menú vamos a ir enseñando cada una de las distintas funcionalidades de nuestra aplicación. Además, iremos enseñándolas en el orden lógico que se debe seguir a la hora de usarla.

4.2. Creación del modelo

Esta funcionalidad consiste en crear un nuevo modelo con las características que el usuario considere oportunas.

Cuando queremos crear un modelo, lo primero que debemos hacer es seleccionar los datos con los que queremos entrenar nuestras técnicas de inteligencia artificial.

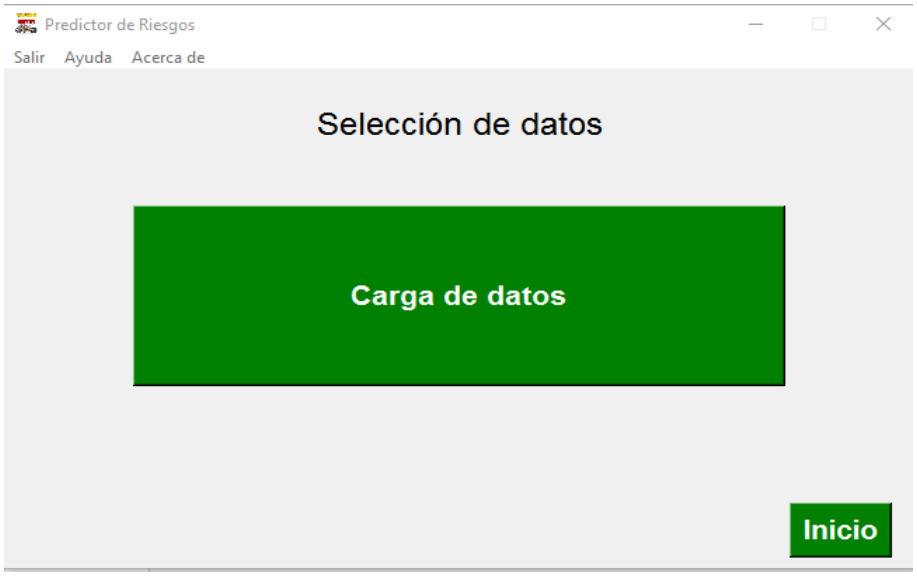


Figura 4.6: Ventana carga de datos.

Una vez que hemos seleccionado los datos a cargar, la aplicación nos preguntará acerca de las características que deseamos tener en cuenta para el modelo.

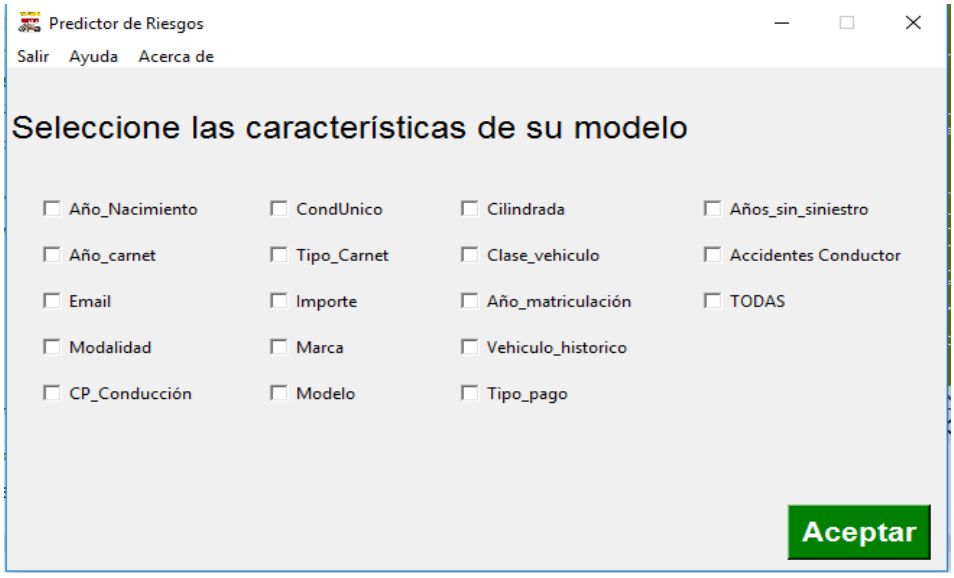


Figura 4.7: Ventana para elegir las características del modelo.

Por otro lado, para los perfiles mas expertos, nuestra aplicación permite la configuración manual de algunos de los parámetros fundamentales de una red neuronal. Es decir, el usuario podrá modificar con facilidad tanto el número de capas que desea en su red neuronal como el número de nodos en cada una de ellas.

En caso de que el usuario sea menos experimentado podrá obviar este apartado y entonces la red neuronal cogerá la configuración por defecto, tres capas de 100 nodos en cada una de ellas.

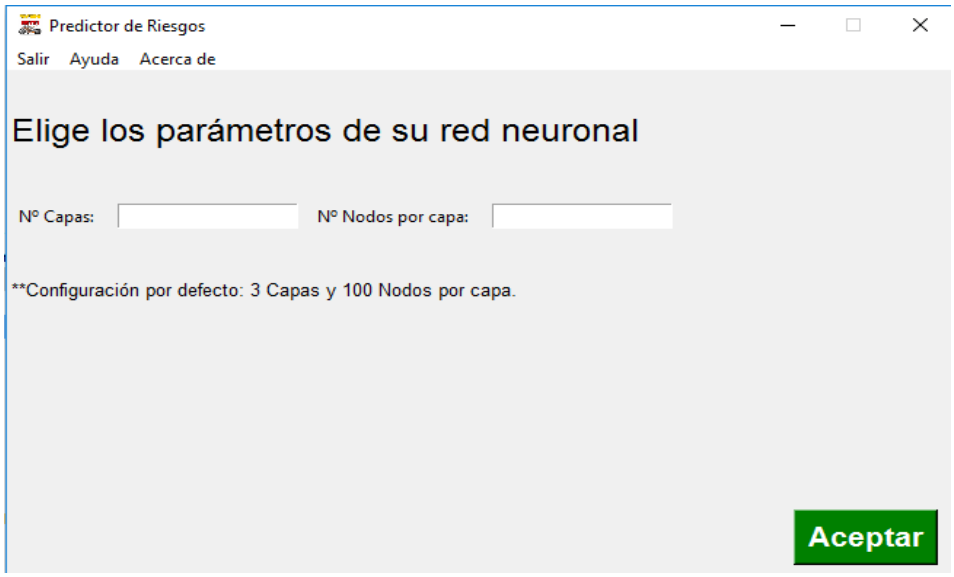


Figura 4.8: Ventana para modificar los parámetros de la red neuronal.

Una vez elegidas tanto las características del modelo que deseamos como los parámetros de la red neuronal, nos saldrá una ventana de confirmación informándonos acerca de las características que tendrá el modelo. Si confirmamos con el botón “Crear Modelo”, estaremos creando nuestro nuevo modelo.

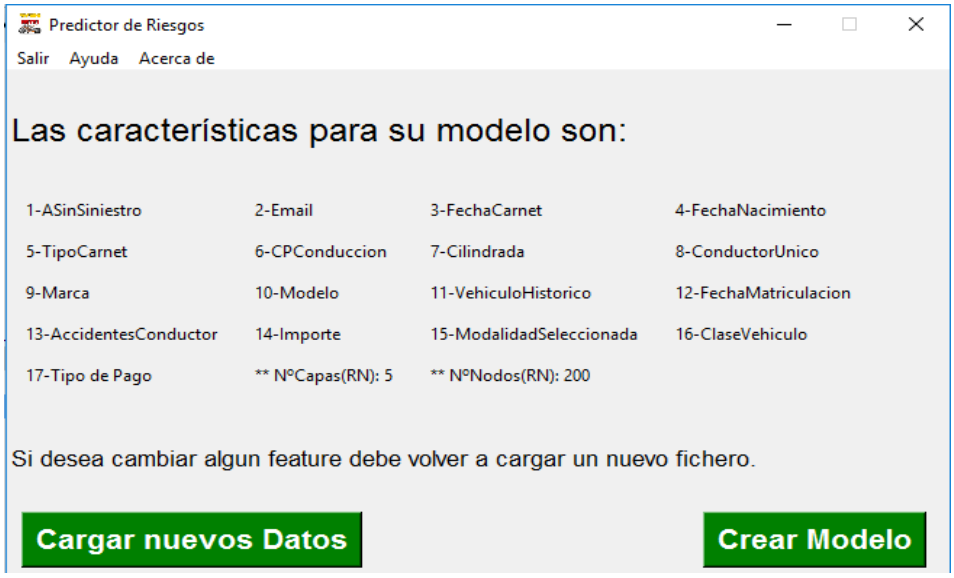


Figura 4.9: Ventana de confirmación.

Una vez seleccionado el botón de crear modelo, nos aparecerá una ventana donde aparecerán los distintos resultados que han conseguido nuestros clasificadores.

Figura 4.10: Ventana de resultados al crear el modelo.

Si dichos resultados no cumplen nuestras necesidades, podremos retroceder y crear otro modelo con características diferentes. Este proceso debe hacerse varias veces hasta conseguir unos resultados que cuadren con el objetivo que se está buscando.

En caso contrario, si los resultados obtenidos cuadran con el objetivo, podremos guardar el modelo para en un momento determinado, cargar el modelo y atender a clientes con él.

Cuando guardamos el modelo, si todo ha ido bien, nos debe salir una ventana emergente indicándonos que la operación ha sido un éxito.

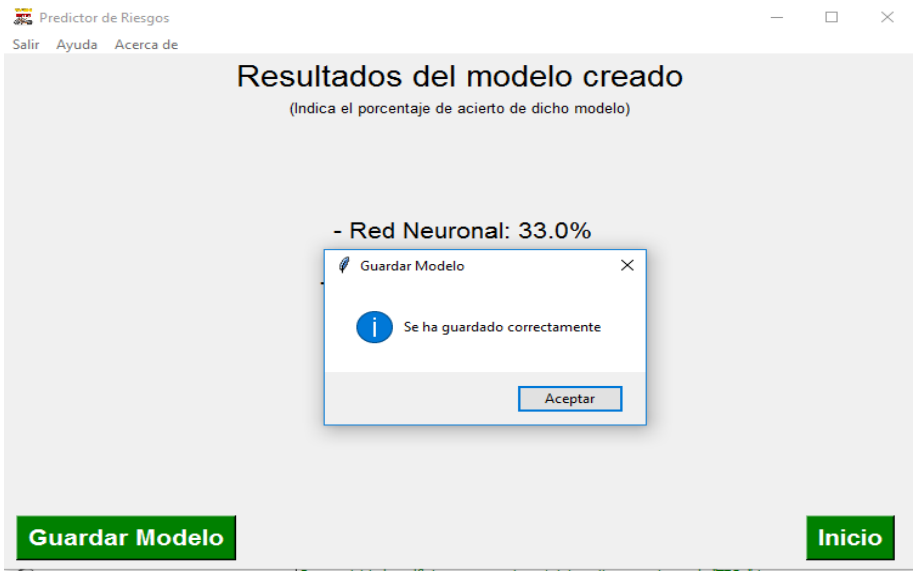


Figura 4.11: Ventana emergente de éxito al guardar un modelo.

Al guardar el modelo, en la ruta donde se esté ejecutando el *script* se nos guardarán los diferentes modelos (uno por cada uno de los clasificadores) y un archivo .txt donde podremos comprobar la información que constituye dicho modelo. Dichos archivos estarán datados con el fin de que sean fáciles de encontrar.

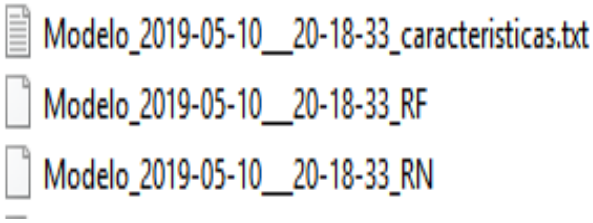


Figura 4.12: Archivos guardados.

Tras todo esto, ya tenemos un modelo totalmente creado a gusto del usuario y listo para ser cargado por parte de la aplicación.

4.3. Carga del modelo

Cuando usamos la funcionalidad de cargar el modelo, la aplicación nos pedirá varios archivos, uno por cada clasificador (Se han generado al crear modelo).

Si todo ha ido bien, nos aparecerá una ventana emergente informándonos que se ha cargado correctamente.

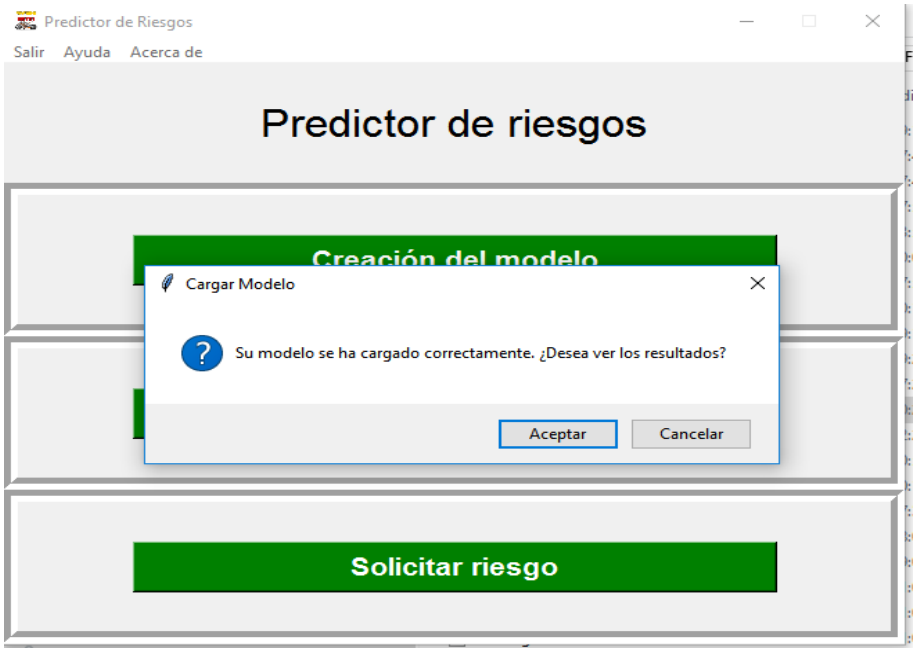


Figura 4.13: Ventana emergente de éxito al cargar un modelo.

Al pulsar en la ventana emergente, nos llevará a una nueva ventana donde podemos ver los resultados obtenidos por parte de los modelos cargados.

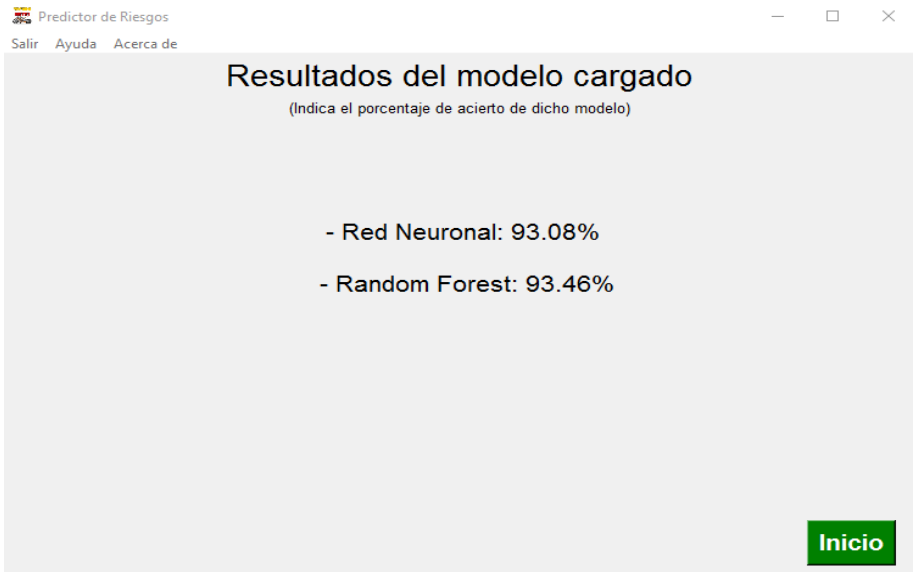


Figura 4.14: Ventana donde se muestran los resultados de un modelo cargado.

Una vez que hemos cargado un modelo, la aplicación está lista para poder solicitar el riesgo de un nuevo cliente. Por ello, en nuestra ventana inicial se nos habilitará la funcionalidad de Solicitar riesgo a diferencia de como estaba en la figura 4.1 al principio del todo.

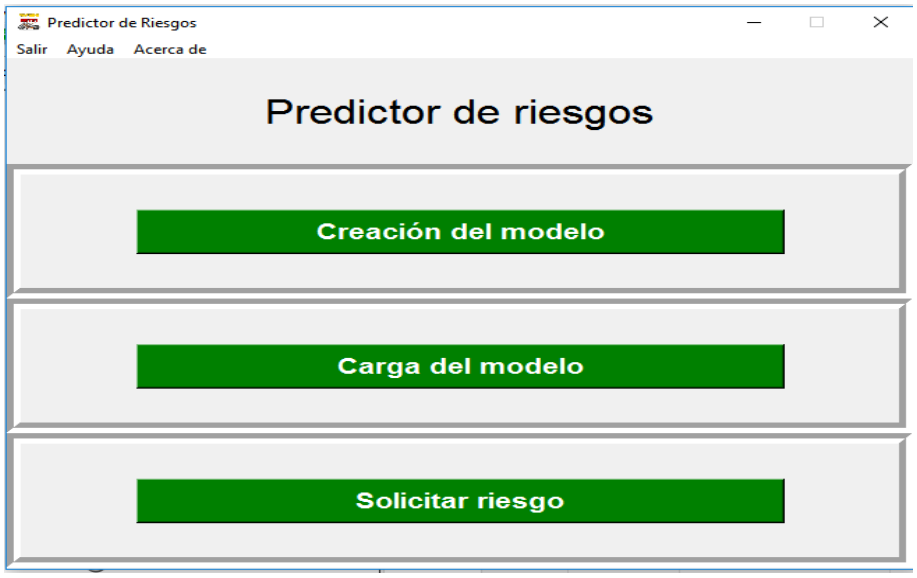


Figura 4.15: Ventana inicial con solicitar riesgo habilitado.

Ademas, nuestra aplicación genera un fichero .txt con un histórico de los modelos cargados. De esta manera, podemos saber cuando hemos cargado cada uno de los modelos y la ruta donde están.

A continuación, en la figura 4.16 mostramos el contenido que tendría el log histórico.

```
Fecha: 2019_05_10 a las 21:40:03
Ruta Modelo (RN) Cargado: C:/Users/dsanchar/Modelo_2019-05-10__20-18-33_RN
Ruta Modelo (RF) Cargado: C:/Users/dsanchar/Modelo_2019-05-10__20-18-33_RF
Ruta Lista características Cargada: C:/Users/dsanchar/Modelo_2019-05-10__20-18-33_caracteristicas.txt

Fecha: 2019_05_10 a las 21:40:11
Ruta Modelo (RN) Cargado: C:/Users/dsanchar/Modelo_2019-05-10__20-18-33_RN
Ruta Modelo (RF) Cargado: C:/Users/dsanchar/Modelo_2019-05-10__20-18-33_RF
Ruta Lista características Cargada: C:/Users/dsanchar/Modelo_2019-05-10__20-18-33_caracteristicas.txt
```

Figura 4.16: Fichero log histórico de modelos cargados.

4.4. Solicitar Riesgo

Esta funcionalidad sirve para generar la predicción del riesgo que tendrá un nuevo cliente. Para poder utilizar esta funcionalidad, debemos haber cargado un modelo en algún momento (Aunque fuese en otra sesión).

El último modelo cargado siempre será la base a la hora de generar las predicciones para el nuevo cliente.

Si solicitamos riesgo, lo primero que nos aparecerá es la ventana de la figura 4.17, donde podemos ver el último modelo cargado y las características del mismo.

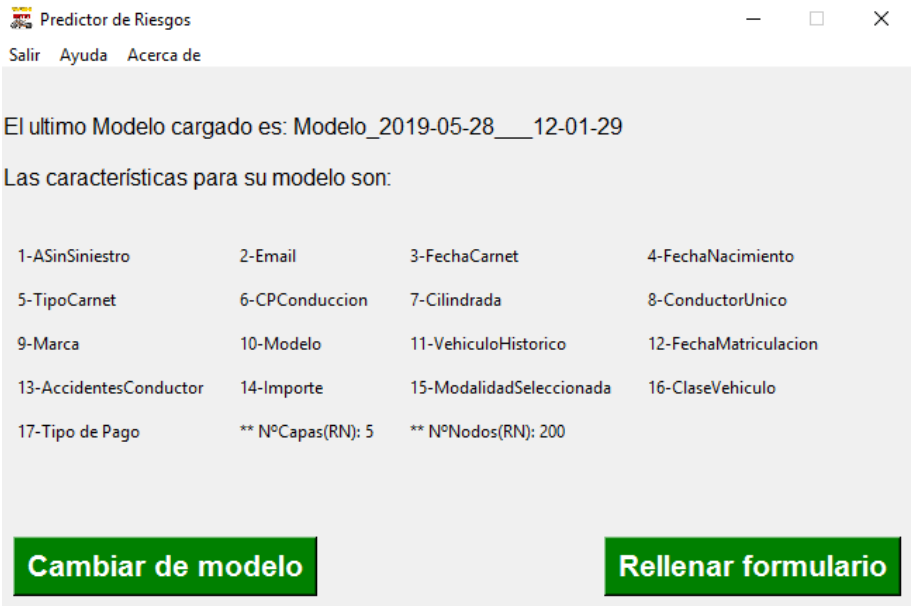


Figura 4.17: Ventana de información al solicitar riesgo.

En caso de que este no sea el modelo deseado, podemos cambiarlo de dos maneras distintas: bien cargando otro modelo ya creado o bien creando uno nuevo. En cambio, si nos gusta el modelo y queremos utilizarlo para solicitar el riesgo de un nuevo cliente, el usuario deberá rellenar un formulario con los datos del cliente.

Dicho formulario consta de dos partes y preguntamos todo tipo de datos necesarios para hacer una buena predicción.

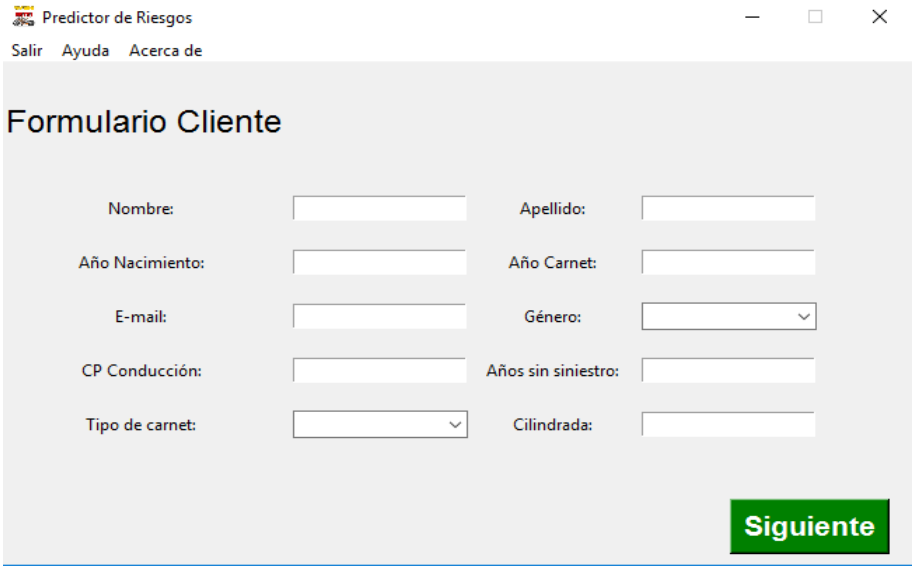


Figura 4.18: Ventana del formulario(1ªParte).

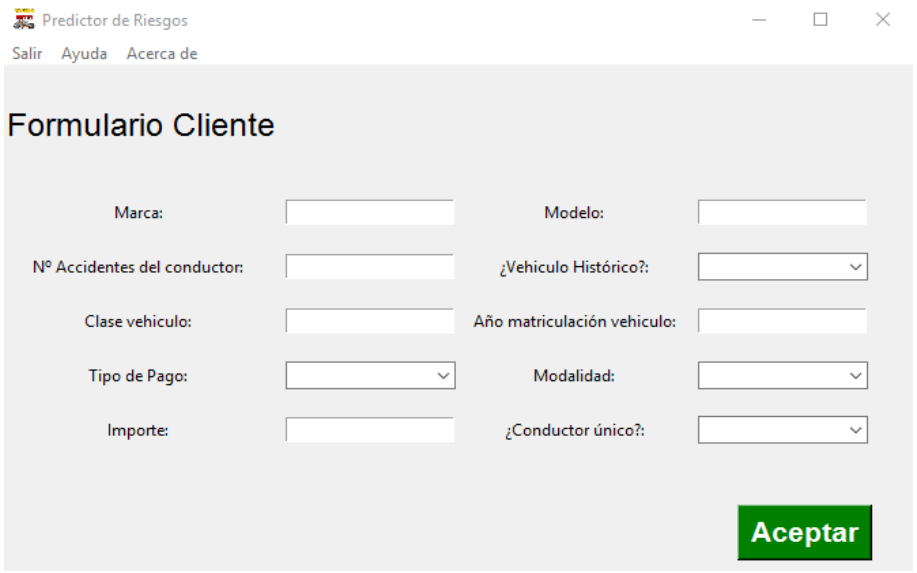


Figura 4.19: Ventana del formulario(2ªParte).

Una vez tengamos el formulario relleno debemos aceptar y entonces veremos la ventana de la figura 4.20 mientras se procesan los resultados.

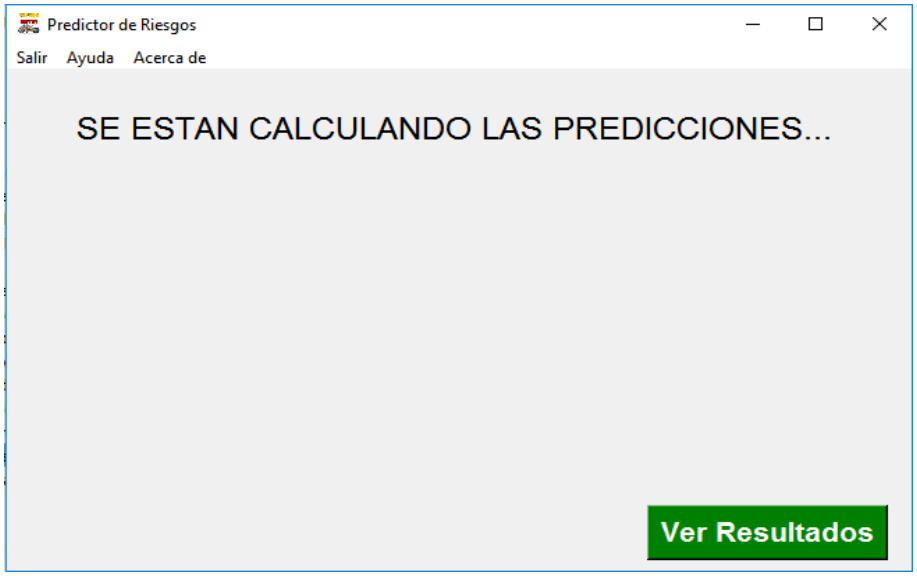


Figura 4.20: Ventana de espera.

Una vez ya tengamos los resultados podremos verlos de la misma manera que muestra la figura 4.21.

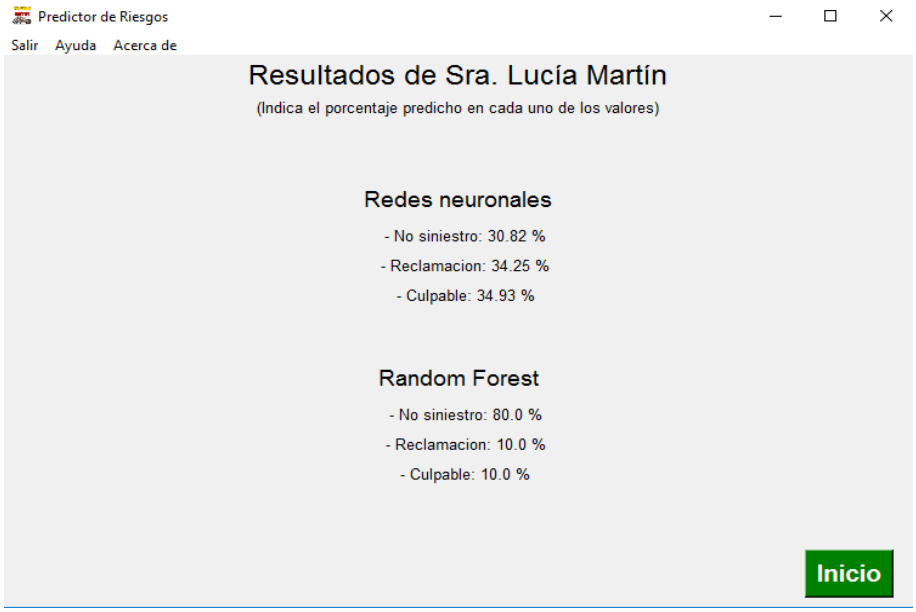


Figura 4.21: Ventana de resultados de solicitar riesgo.

Con todo este paseo tecnológico por nuestra aplicación, habremos calculado el riesgo que tiene un nuevo cliente gracias a las técnicas de inteligencia artificial creadas en el back-end de la aplicación, las cuales explicaremos en el siguiente capítulo.

Capítulo 5

Back-end de la aplicación

En este capítulo nos centramos en explicar el funcionamiento interno de la aplicación que hace posible las predicciones.

En primer lugar, explicaremos para cada una de las técnicas su evolución histórica y en que consisten. Por último, explicaremos como hemos conseguido implementarlas en nuestra aplicación.

5.1. Redes Neuronales

La idea inicial que que finalmente acabó siendo las redes neuronales nació de los expertos Warren McCulloch y Walter Pitts cerca del año 1943. Estos expertos crearon un primer modelo llamado "lógica umbral", el cual se basaba fundamentalmente en matemáticas e informática. [7]

Entre las décadas de 1950 y 1960 el científico Frank Rosenblatt, creó el perceptron, la unidad desde donde nacería y se potenciarían las redes neuronales artificiales. Un perceptron toma varias entradas binarias y produce una sola salida binaria. Además, Rosenblatt introduce el concepto de pesos.

Ya en el año 1965 se creó el perceptron multicapa que como se puede imaginar,es una ampliación del perceptron de una única neurona a más de una. En ese momento, es cuando aparecen los conceptos de capas de entrada, capas ocultas y capa de salida los cuales explicaremos mas adelante.

La década de los 80 es muy destacada en las redes neuronales ya que se empieza a aplicar en el mundo de la inteligencia artificial, más concretamente en el aprendizaje automático. Se empiezan a poder entrenar gracias al algoritmo *backpropagation* y se comienza a utilizar frecuentemente en problemas de clasificación hasta que la falta de cómputo hace que el avance quede algo paralelizado.

A partir del 2006, se logra superar la barrera y gracias al crecimiento en el poder de computación y de las nuevas ideas se logra entrenar cientos de capas jerárquicas que conforman y potencian lo que se ha denominado Deep learning.

A día de hoy, existen nuevos conocimientos acerca de las neuronas humanas biológicas en las cuales se está redescubriendo su funcionamiento y se está produciendo una nueva revolución, ya que según parece es distinto a lo que pensábamos. Esto parece ser el inicio de una nueva época la que según los expertos se denomina el machine learning y la inteligencia artificial.

Una red neuronal se trata de un conjunto de neuronas conectadas mediante enlaces, las cuales trabajan para un fin común. A medida que pasa el tiempo, con la experiencia, las neuronas van creando y reforzando ciertas conexiones entre ellas por si solas. Es decir, van "aprendiendo" cuáles son las mejores conexiones de manera totalmente autónoma. [8]

El elemento más básico de computación de una neurona se le llama habitualmente "nodo"los cuales cada uno de ellos reciben un *input* en forma de datos externos. Cada uno de estos nodos recibe un peso el cual se va modificando en el proceso de aprendizaje de la red neuronal, dando más peso a aquellas conexiones que sean mejores. [9]

EL objetivo de una red neuronal siempre es convertir la información de entrada en unos valores de salida.

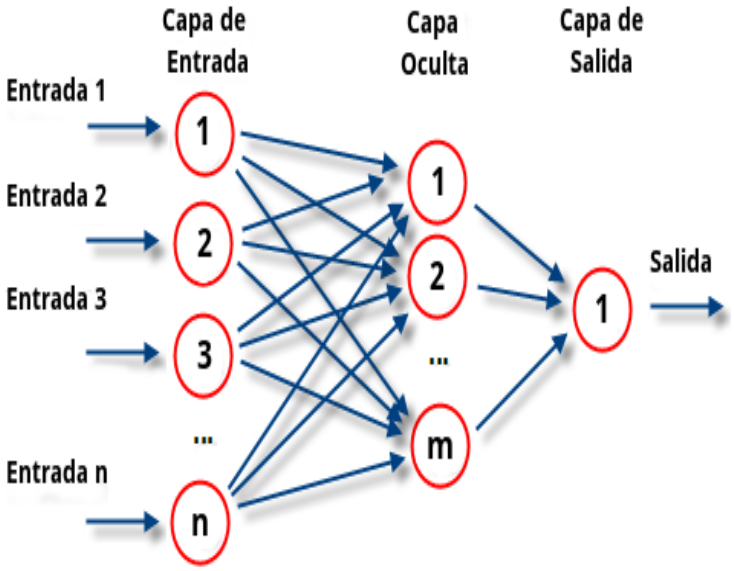


Figura 5.1: Esquema de una red neuronal.

A la hora de implementar una red neuronal hay que tener claro que características queremos que tenga. A continuación explicamos cuáles son los mas importantes:

1. Capas de entrada: Capas que reciben directamente los datos del problema.
2. Capas ocultas: Son aquellas que no tienen una conexión directa con el entorno y se encuentran entre las capas de entrada y la de salida. Una red neuronal puede tener tantas capas ocultas como se desee, incluso ninguna. Cuanto mayor sea la cantidad de capas, mayor será el número de maneras distintas en las que se están interconectando las neuronas de la red ya que la salida de una capa será la entrada de la otra.
3. Capa de salida: Capa que da la respuesta final tras todo el proceso.
4. Número de nodos por capa: Son los distintos *inputs* que tiene cada una de las capas. A mayor número de nodos, mayor interconectividad con el resto de capas.

Como es lógico, si aumentamos demasiado el número de capas y/o de nodos, el tiempo de ejecución crece a pasos agigantados. Por ello, hay que buscar siempre un término medio entre la mejor solución y los costes de tiempo y de computación.

En la modelización con redes neuronales hay dos fases principales:

- Fase de entrenamiento: Fase en la que se usa un subconjunto de los datos con el fin de que nuestra red neuronal pueda determinar los pesos que definen el modelo. El objetivo es minimizar el error cometido entre la salida obtenida por la red neuronal y la salida deseada. En esta fase hay que tener especial cuidado para que nuestro modelo no se ajuste demasiado a las particularidades del subconjunto de datos que hemos dado para entrenar, ya que si esto ocurre, nuestro modelo no será capaz de generalizar su aprendizaje a nuevos casos (sobreajuste) y por tanto, no será útil.
- Fase de prueba: En esta fase es cuando veremos si nuestra red neuronal esta funcionando de manera correcta o no. Para comprobar que nuestro modelo no esta sobreajustado es muy aconsejable utilizar un subconjunto de datos distinto al del entrenamiento.

A continuación exponemos las ventajas e inconvenientes más importantes de este clasificador:

Ventajas

- Flexibilidad ya que puede manejar cambios (no muy importantes) en la información de entrada.
- Tolerancia a fallos.

Inconvenientes

- Complejidad de aprendizaje.
- No permite interpretar lo que se ha aprendido sino que la red neuronal proporciona una salida y es el programador quien debe interpretarla.
- Se necesita una gran muestra de datos para que funcione de manera óptima.

5.2. Random forest

El algoritmo denominado Random forest fue creado por Leo Breiman y Adele Cutler. El nombre de Random forest fue puesto porque era la fabrica donde ellos trabajaban. El termino apareció por primera en 1995 por Tin Kam Ho. [11]

Este método es una combinación de la idea inicial de un método denominado *bagging* que tenía Breiman junto con la idea de Ho sobre la selección aleatoria de atributos.

La introducción de bosques aleatorios propiamente dichos se realizó por primera vez en un artículo de Leo Breiman. En este documento se habla de un método para construir un bosque de árboles no correlacionados. Además, este documento combina varias ideas, algunas ya conocidas y otras novedosas, que forman la base de la práctica moderna de bosques aleatorios, en particular, el Random forest.

Este algoritmo es muy útil tanto para problemas de regresión como de clasificación. Para muchos expertos es uno de los mas importantes en los problemas de ciencia de datos.

La idea principal del Random forest consiste en generar un número importante de árboles, donde cada uno de ellos vota la clase a la que pertenece el ejemplo nuevo. La clase que se le asigne al nuevo ejemplo será aquella que gane en la votación por mayoría de todos los árboles.

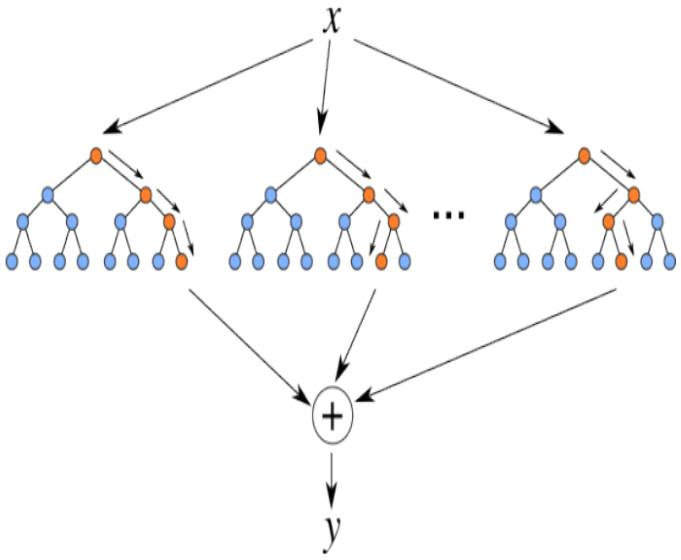


Figura 5.2: Esquema del Random forest.

El algoritmo de Random forest sigue el siguiente proceso:

- Selección de individuos de manera aleatoria para crear diferentes conjunto de datos.
- Cada conjunto de datos creará un árbol de decisión, obteniendo así un conjunto de arboles. Cada árbol es distinto ya que cada subconjunto contiene diferentes individuos y diferentes características.
- Al crear los arboles se eligen características al azar en cada nodo del árbol, dejando crecer el árbol en profundidad (es decir, sin podar).
- La predicción se genera usando la idea del "voto mayoritario". Se clasificará como "positivo"solo en el caso de que la mayoría de los árboles hayan predicho la observación como positiva.

A continuación exponemos las ventajas e inconvenientes más importantes de este clasificador:

Ventajas

- No se necesita mucha preparación de los datos.
- El algoritmo puede manejar una gran cantidad de características e identificar las más significativas por sí solo.

Inconvenientes

- Poco control en lo que hace el modelo internamente.
- Poca posibilidad de interpretación de los resultados.

Cabe destacar, que el resultado que muestran los clasificadores no deja de ser pura probabilidad y que por tanto no tiene porque ser totalmente correcta.

5.3. Nuestra aplicación

En está sección, vamos a contar con máximo detalle como hemos implementado la parte back-end. Comenzaremos explicando como hemos realizado el tratamiento inicial de los datos y finalizaremos explicando la implementación propia de los clasificadores.

5.3.1. Lectura, limpieza y tratamiento de datos

Antes de entrar en la propia materia, vamos a intentar dejar claro el concepto de *data frame* ya que va a ser muy importante.

Data Frame

Un *data frame*, es el nombre que la biblioteca panda ha dado a un marco de datos. Este marco de datos es una matriz donde las filas son los valores de los datos y las columnas las distintas características. Este conjunto de datos será la entrada de las distintas técnicas y las predicciones saldrán en base a él.

Por ello, debe tratarse correctamente para poder facilitar el trabajo lo máximo posible a las distintas técnicas de predicción.

Estas son las principales operaciones de tratamiento que hay realizar para tener un buen data frame:

1. Limpieza de datos y rellenado de huecos. Preparamos los datos de manera correcta para poderlos tratar.
2. Creación y tratamiento de *features* (Características) a partir de los datos.
3. Eliminación de features no importantes (Features extraction)

Estas fases, sin duda ha sido uno de los mayores quebraderos de cabeza que hemos tenido continuamente a la hora de realizar el proyecto.

Esta parte del proyecto es la que posteriormente, cuando expliquemos los clasificadores denominaremos “preprocesado”. Es una parte importantísima para cualquier software de aprendizaje automático ya que sin ella los datos no estarían bien tratados, y por tanto, los clasificadores jamás podrán funcionar correctamente.

Sin mas dilación, comenzamos a explicar como lo hemos implementado:

Para la limpieza y el tratamiento de datos principalmente hemos usado la biblioteca por excelencia para este tipo de trabajos como es Panda [4].

Lo primero que hay que realizar, es cargar el conjunto de datos que tenemos. En nuestro caso, como ya dijimos al principio de este documento, los datos nos los dieron en formato Excell. Por ello, utilizamos la siguiente función de panda.

```
data = pd.read_excel("DatosSeguros_Eq.xlsx")
```

Figura 5.3: Lectura de datos.

Normalmente los datos que se manejan en problemas reales no están limpios y por ello, al igual que tuvimos que hacer nosotros, es frecuente tener que rellenar con 0's aquellos datos que vienen con NULL o vacíos.

```
clean_data = data.fillna(0)#Rellenamos con 0's
```

Figura 5.4: Relleno de datos.

A la hora de decidir que columnas serían las adecuadas y cuáles son despreciables para nuestros clasificadores lo primero que hicimos fue juntarnos con los directores del proyecto y decidirlo. Tras varias conversaciones, llegamos a la conclusión que ciertas columnas no nos iban aportar nada a la hora de predecir y es por ello que decidimos eliminarlas. Las columnas que consideramos importantes para predecir estarán disponibles en la interfaz y debe ser el experto quien debe crear los modelos en base a las que él desee.

```
#Eliminamos las columnas que no nos interesan.
dropable = ['ActualmenteAsegurado','SiniestrosNoResponsable3YA',
clean_data = clean_data.drop(dropable, axis = 1) #Axis=1 para qu

clean_data.columns
#Mostramos como queda con los features que sí nos interesa
```

Figura 5.5: Eliminación de columnas.

Los clasificadores, no entienden de letras sino que necesitan que el data frame sea complemente numérico. Por ello, en varias de las columnas es importante hacer un hash para pasar determinadas cadenas a valores numéricos.

Podemos distinguir dos grandes tipos de columnas (características):

1. Columnas donde no existe gran variedad de valores y se suelen repetir. Para ellas, nuestra manera de operar es hacer un map y pasar cada cadena a un valor numérico.
2. Columnas donde existe multitud de valores. En estas columnas, nuestra manera de operar es hacer un ranking para coger los grupos de valores que más se repitan y los que no se repitan agruparlos en un grupo denominado "otros".

Columnas sin variedad

A continuación, mostramos un ejemplo de cómo se realiza este hash. Lo que hacemos es cambiar determinadas cadenas que se repiten a un número entero.

En el ejemplo de la figura 5.6 vemos como transformamos la columna "AsistenciaViajePremium" con el siguiente procedimiento: donde aparezca "PagoAnual"pasará a ser un 1, mientras que donde aparezca "PagoSemestral"pasará a ser un 2.

```
clean_data['AsistenciaEnViajePremium.1'].map({'PagoAnual':1, 'PagoSemestral':2,
```

Figura 5.6: Cambio de cadena a valor numérico (Columnas sin variedad).

Este proceso no solo se ha hecho con esta característica sino que también lo hemos tenido que utilizar con otras.

Columnas con variedad

A continuación, vamos a explicar como hemos realizado el mapeo en una característica donde existen multitud de valores.

Para explicarlo, nos basaremos en el mapeo que hemos realizado a la característica "marca". Si bien es cierto que existen muchos valores de marca en el conjunto de datos, hay determinadas marcas que se repiten frecuentemente. Por ello, lo que hicimos fue seleccionar aquellas marcas que se repetían frecuentemente y asignar cada una de ellas a un número diferente. Para el resto de marcas, las cuales salían en los datos de manera muy puntual las asignamos a un mismo número a todas ellas. Este número, se podría identificar como un grupo denominado "otros".

En el ejemplo de la figura 5.7 podemos ver como lo hemos implementado.

```
def mapMarcas(nombre):

        if(nombre == 'YAMAHA'):
            result = 1

        elif(nombre == 'APRILIA'):
            result = 2

        elif(nombre == 'KAWASAKI'):
            result = 3

        elif(nombre == 'PIAGGIO-VESPA'):
            result = 4

        elif(nombre == 'SUZUKI'):
            result = 5

        elif(nombre == 'HONDA'):
            result = 6

        elif(nombre == 'RIEJU'):
            result = 7

        elif(nombre == 'BMW'):
            result = 8

        else:
            result = 9

        return result
```

Figura 5.7: Cambio de cadena a valor numérico (Columnas con variedad).

Este proceso no solo se ha hecho con esta característica sino que también lo hemos utilizado con otras características como son modelo de coche, correo electrónico del cliente, etc.

Estos procesos es importante tenerlos claros ya que hay que hacerlos repetidas veces para poder ofrecer a los clasificadores un data frame totalmente numérico.

Para el tratamiento de las columnas con fechas como pueden ser fecha de nacimiento, fecha de carnet de conducir, etc. lo que hacemos es quedarnos solo con el año ya que el mes y el día consideramos que no es suficientemente relevante a la hora de predecir. Por ello, en las fechas tan solo nos quedaremos con el año.

```
clean_data['Año_Carnet'] = clean_data.FechaCarnet.dt.year.astype(int)

clean_data['Año_Nacimiento']= clean_data.FechaNacimiento.dt.year.astype(int)

clean_data['Año_Matriculacion']= clean_data.FechaMatriculacion.dt.year.astype(int)
```

Figura 5.8: Tratamiento de fechas.

Por último, comentar que es muy recomendable escalar los datos [16] ya que esto puede hacer que se mejoren notablemente las predicciones ya que esto hace que el conjunto de datos estén en un mismo rango y de esta manera los clasificadores podrán asimilar las semejanzas de manera mucho mas sencilla.

En el conjunto de datos que nos proporcionaron hay una gran variedad de tipos de datos con diversas magnitudes. Después de informarnos sobre el machine learning nos dimos cuenta que es una práctica que se recomienda repetidamente porque puede tener un gran impacto en ciertos algoritmos.

El impacto se debe a que, como hemos comentado, la gran variedad de magnitud en las características hace que el peso que tienen en el algoritmo puede ser diferente aunque en realidad tenga que ser el mismo.

Por ejemplo, si tuviéramos dos características en las cuales se representa el peso de objetos y en una se representara en kilogramos y otra en gramos el algoritmo podría asignarle un peso, en el predictor, mucho más grande a mil gramos que a un kilogramo cuando el valor real es el mismo.

Entonces, lo que hace el escalado es unificar todos los valores para que no ocurran las situaciones como la que hemos puesto de ejemplo.

La manera de escalar un data frame es la que se puede ver en la figura 5.9.

```
def crearModelo(clean_data,num_nodos,num_capas):

    X_train, X_test, y_train, y_test = split(clean_data)

    scaler = StandardScaler()
    scaler.fit(X_train)

    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)

    tupla = capasRed(str(num_nodos),int(num_capas))

    clf = fit(X_train,y_train,tupla)

    return clf
```

Figura 5.9: Escalar datos.

5.3.2. Clasificadores

En esta sección, empezaremos contando las distintas fases por las que debe pasar cualquier clasificador y acabaremos contando como hemos implementado los clasificadores, en nuestro caso: Redes neuronales y Random forest.

Lo primero que debemos saber es que los clasificadores consisten en que dado unos parámetros de entrada (el data frame tratado), devuelven un resultado totalmente probabilístico. Para ello, los algoritmos deben pasar por distintas fases para intentar garantizar la mayor eficacia posible.

El tratamiento de estos clasificadores consta de 3 fases principales más una fase de preprocesado. A continuación, las explicamos:

- *Preprocesado*

Consiste en el tratamiento del data frame descrito en la sección anterior. En nuestro caso, tuvimos que trabajar mucho en este apartado ya que los datos que nos proporcionaron no estaban como necesitábamos.

- *1º Fase: Entrenamiento*

En esta fase utilizaremos un 70 % de los datos obtenidos. En ella, entrenamos el clasificador para que pueda aprender a predecir de manera correcta.

Esta fase consta de dos principales partes:

1. Configuración del clasificador
2. Obtención del modelo

Los resultados obtenidos en esta fase tendrán una probabilidad de ser correcta pseudoaleatoria. Debemos tener en cuenta que soluciones erróneas también aportaran información útil.

- *2º Fase: Validación*

En esta fase utilizaremos un 15 % de los datos obtenidos. Seleccionamos de ellos cual va a ser la táctica a utilizar para que el clasificador funcione de la mejor manera posible.

- *3º Fase: Test*

En esta fase utilizaremos el 15 % de los datos que nos faltaban por utilizar. En este momento lo que hacemos es poner a prueba nuestro clasificador para ver si predice de manera correcta.

A continuación, exponemos como hemos implementado cada uno de los clasificadores:

Red neuronal

Para implementar la red neuronal hemos utilizado la biblioteca de Python *sklearn* la cual es la biblioteca por excelencia para la mayoría de los expertos.

Lo primero que tenemos que tener claro es saber que columna queremos predecir. En nuestro caso, se trata de la columna "siniestro".

```
X = clean_data.drop("Tipo siniestro",1)
y = clean_data["Tipo siniestro"]
```

Figura 5.10: Columna a predecir.

Al predecir esta columna, podremos saber los porcentajes que tiene un nuevo cliente en cada de uno de los siguientes valores:

- Sin Siniestro
- Reclamación
- Culpa

Para un uso correcto de la red neuronal, lo primero que debemos hacer es repartir sus datos para las distintas fases (entrenar, test) como hemos explicado anteriormente. En nuestro caso, vamos a utilizar un 30 % de los datos para el test pero esto puede variar. Para ver como se hace esta división debemos ver el ejemplo de la figura 5.11

```
X_train, X_test, y_train, y_test = train_test_split(clean_data.drop("Tipo siniestro",1), clean_data["Tipo siniestro"],
                                                    test_size=0.3,
                                                    random_state=0)
```

Figura 5.11: División de los datos.

Una vez que ya hemos repartido el conjunto de datos, debemos crear y entrenar nuestro clasificador. En primer lugar, nos creamos el tipo de clasificador que queremos. En este caso, una red neuronal, para ello debemos invocar al método *MLPClassifier()* [10].

Posteriormente con el método *fit()* [17] entrenamos nuestra red para que sea capaz de predecir con buenos resultados.

```
mlp = MLPClassifier()
mlp.fit(X_train,y_train)
```

Figura 5.12: Entrenamiento de la red neuronal.

Para saber que precisión nos proporciona nuestra red neuronal, es decir, para saber que porcentaje de acierto tiene nuestra red debemos llamar al método *score()* de la biblioteca sklearn.

Es muy común mirar la precisión que ha tenido nuestra red neuronal tanto a la hora del test (datos desconocidos) como a la hora del entrenamiento (datos conocidos). En la figura 5.13 podemos ver como se implementa.

```
print("Training set score: %f" % mlp.score(X_train, y_train))
print("Test set score: %f" % mlp.score(X_test, y_test))
```

Figura 5.13: Precisiones de test y de entrenamiento.

Sin duda, el porcentaje que más nos debe influir a la hora de ver si la red neuronal esta entrenada correctamente y ver si predice bien es la precisión del test ya que es la que evalúa si la red neuronal se adapta bien a nuevos datos.

A veces, una red neuronal predice muy bien para los datos de entrenamiento pero en cambio, para datos nuevos predice mucho peor. Este problema ocurre cuando se ha producido un sobreajuste y se debe solucionar.

Por ultimo, lo que debemos hacer es llamar al método de sklearn *predict()*. Este método será el que nos proporcione la información que estamos buscando, es decir, los porcentajes de cada uno de los valores posibles de la columna a predecir.

```
predictions = mlp.predict(X_test)
```

Figura 5.14: Método para predecir.

Es muy útil, para comprobar los resultados, analizarlos y poder hacer pruebas tener en cuenta la matriz de confusión [15], la cual nos da información para ver que tal funciona nuestro clasificador. Para ver como se implementa una matriz de confusión debemos ver la figura 5.15.

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

Figura 5.15: Matriz de confusión.

Cabe destacar, que una red neuronal no es algo rápido de implementar ya que existen diversos parámetros de la red como son el numero de capas, algoritmo que usa, etc. que hay que ir variando e ir probando hasta encontrar el mejor funcionamiento de la misma.

Random forest

El clasificador Random forest se crea de manera muy parecida al proceso de creación de la red neuronal y se debe pasar por los mismos pasos mencionados en la sección anterior.

Lo único que varía en este caso es el tipo de clasificador a crear. Mientras que en la sección anterior creábamos una red neuronal, ahora debemos crear un clasificador Random forest. Para ello, debemos implementarlo como indica la figura 5.16

```
clf = RandomForestClassifier()
```

Figura 5.16: Creación Random forest.

Esto no significa que con saber un clasificador ya no necesites formarte en los demás ya que la dificultad de estos clasificadores reside en la configuración de los mismos la cual es totalmente independiente.

Capítulo 6

Análisis de los resultados

A ser sinceros, lo cierto es que no hemos conseguido los resultados esperados para nuestros clasificadores.

Las posibilidades que hemos pensado que ocasionan una mala predicción de nuestros clasificadores son las siguientes:

- La muestra de datos es insuficiente.
- Los datos no tienen suficiente correlación para que el clasificador saque conclusiones.
- Nuestro análisis de los datos y/o La parametrización de los clasificadores no ha sido del todo correcta.

Para ver si realmente es culpa nuestra o son los datos de entrada, vamos a realizar unas pruebas donde vamos a predecir con un conjunto de datos sintéticos por nosotros mismos. Además, estos datos serán mucho más sencillos y solo diferenciaremos si el cliente tendría accidente o no.

Ejecución de las pruebas con datos sintéticos

Estas pruebas han sido implementadas con una red neuronal de diez capas y con cien nodos por cada una de ellas y por el clasificador Random forest.

Se ha creado un conjunto de datos de mil individuos. La manera de operar será empezar con pocas columnas e ir añadiendo poco a poco para poder observar que cambios va sufriendo la predicción.

A la hora de entrenar los clasificadores, lo haremos de dos formas:

1. Con datos sin escalar: Los datos no sufren ningún cambio a los reales.
2. Con datos escalados: Los datos sufren una transformación para que todos estén en un mismo rango de valores. Es una práctica que se recomienda y que suele mejorar el rendimiento del clasificador. En nuestro caso real no había mucha mejora, pero podremos observar que si es beneficioso para nuestros datos.

De los mil individuos que hemos creado, haremos el siguiente reparto:

- 500 individuos tendrán entre 25 y 28 años con un coche de 20.000 a 30.000 kW de potencia. Todos ellos, diremos que tendrán accidente.
- 500 individuos tendrán entre 35 y 70 años con un coche de 500 a 10.000 kW de potencia. Ninguno de estos individuos sufrirá un accidente.

Estos datos serán constantes a lo largo de las pruebas e iremos añadiendo más columnas según vayamos avanzando siempre y cuando vayan funcionando correctamente.

Las columnas que vayamos añadiendo será para complicar la predicción ya que los datos serán más aleatorios y tendrán menos relación.

Pruebas Red neuronal

A continuación, mostraremos los resultados obtenidos en cada una de las pruebas:

• Prueba 1

Tan solo usamos las dos columnas explicadas anteriormente.

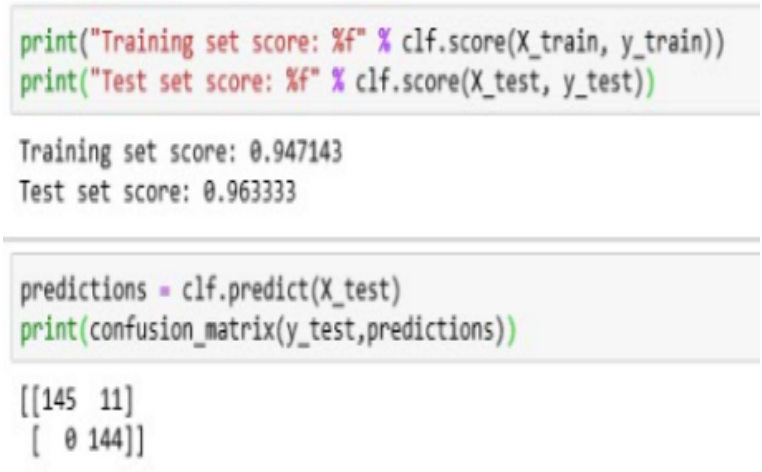


Figura 6.1: Primera prueba (RN) con los datos sin escalar.

Como podemos ver en la figura 6.1 los resultados son casi perfectos y tan solo hay 11 individuos que nuestra red neuronal predice mal.

En cambio, cuando escalamos los datos, los resultados que nos dan son totalmente perfectos y no hay un solo individuo que prediga mal.

```
print("Training set score: %f" % clf.score(X_train, y_train))
print("Test set score: %f" % clf.score(X_test, y_test))
```

```
Training set score: 1.000000
Test set score: 1.000000
```

```
predictions = clf.predict(X_test)
print(confusion_matrix(y_test,predictions))
```

```
[[156   0]
 [  0 144]]
```

Figura 6.2: Primera prueba (RN) con los datos escalados.

• Prueba 2

Añadimos la columna género de forma aleatoria a los dos subconjuntos de los datos.

En la figura 6.3 podemos ver que forma tendrá nuestro conjunto de datos.

	Edad	Potencia	Tipo	Genero
0	25	29357	1	1
1	28	23997	1	1
2	28	20497	1	2
3	25	21250	1	1
4	26	25881	1	2

Figura 6.3: Datos de la segunda prueba (RN).

A continuación, mostraremos los resultados obtenidos tanto con los datos escalados como sin escalar. Como ocurría en la primera prueba, con los datos escalados funciona de manera perfecta mientras que cuando los datos están sin escalar funciona bastante bien pero sin llegar a la perfección.

```
print("Training set score: %f" % clf.score(X_train, y_train))
print("Test set score: %f" % clf.score(X_test, y_test))
```

```
Training set score: 0.977143
Test set score: 0.986667
```

```
predictions = clf.predict(X_test)
print(confusion_matrix(y_test,predictions))
```

```
[[152   4]
 [  0 144]]
```

Figura 6.4: Segunda prueba (RN) con los datos sin escalar.

```
print("Training set score: %f" % clf.score(X_train, y_train))
print("Test set score: %f" % clf.score(X_test, y_test))
```

```
Training set score: 1.000000
Test set score: 1.000000
```

```
predictions = clf.predict(X_test)
print(confusion_matrix(y_test,predictions))
```

```
[[156   0]
 [  0 144]]
```

Figura 6.5: Segunda prueba (RN) con los datos escalados.

- Prueba 3

En esta tercera prueba añadiremos la columna marca con valores aleatorios entre 1 y 6 por cada uno de los individuos.

En este caso, ya vemos en la figura 6.6 un claro ejemplo de como si los datos están sin escalar el clasificador comienza a fallar de manera notable dando un numero erróneo de 95 individuos.

```
print("Training set score: %f" % clf.score(X_train, y_train))
print("Test set score: %f" % clf.score(X_test, y_test))
```

```
Training set score: 0.698571
Test set score: 0.683333
```

```
predictions = clf.predict(X_test)
print(confusion_matrix(y_test,predictions))
```

```
[[ 61  95]
 [  0 144]]
```

Figura 6.6: Tercera prueba (RN) con los datos sin escalar.

En cambio, si los datos están escalados todo sigue funcionando perfecto y nuestro clasificador sigue sin cometer ningún error.

```
print("Training set score: %f" % clf.score(X_train, y_train))
print("Test set score: %f" % clf.score(X_test, y_test))
```

```
Training set score: 1.000000
Test set score: 1.000000
```

```
predictions = clf.predict(X_test)
print(confusion_matrix(y_test,predictions))
```

```
[[156   0]
 [  0 144]]
```

Figura 6.7: Tercera prueba (RN) con los datos escalados.

- Prueba 4

En esta última prueba añadiremos la columna modelo con valores aleatorios entre 1 y 10 por cada uno de los individuos.

Como era de esperar, si los datos siguen sin funcionar y al ser mayor el número de columnas, a nuestro clasificador le cuesta más y falla más que en la prueba anterior. En este caso, son 101 los individuos que están mal predichos.

```
print("Training set score: %f" % clf.score(X_train, y_train))
print("Test set score: %f" % clf.score(X_test, y_test))
```

```
Training set score: 0.672857
Test set score: 0.663333
```

```
predictions = clf.predict(X_test)
print(confusion_matrix(y_test,predictions))
```

```
[[ 55 101]
 [  0 144]]
```

Figura 6.8: Cuarta prueba (RN) con los datos sin escalar.

Si miramos los resultados con los datos escalados vemos que siguen funcionando perfectamente lo cual nos hace ver que escalar los datos es algo fundamental y que puede variar los resultados de manera muy drástica.

```
print("Training set score: %f" % clf.score(X_train, y_train))
print("Test set score: %f" % clf.score(X_test, y_test))
```

```
Training set score: 1.000000
Test set score: 1.000000
```

```
predictions = clf.predict(X_test)
print(confusion_matrix(y_test,predictions))
```

```
[[156   0]
 [  0 144]]
```

Figura 6.9: Cuarta prueba (RN) con los datos escalados.

Pruebas Random forest

En primer lugar, realizamos las mismas pruebas hechas para la red neuronal. En la figura 6.10 se puede ver que este clasificador constantemente predecía de manera perfecta.

Prueba	Resultado datos sin escalar	Resultados con escalado
Mil individuos diferenciados por edad y potencia del vehículo	100 %	100 %
Añadimos la característica de género	100 %	100 %
Añadimos la característica de marca	100 %	100 %
Añadimos la característica de modelo	100 %	100 %

Figura 6.10: Pruebas con el clasificador Random forest.

Tras estos resultados, consideramos que los datos no nos daban una idea de con qué tipo de datos empieza el clasificador Random forest a fallar.Por lo tanto, tuvimos que cambiar los datos con los que entrenar pero conservando la misma forma que en las pruebas con la red neuronal.

Decidimos acortar las diferencias que habíamos supuesto en los primeros datos creados. En esta ocasión, la edad de accidente es de 18 a 50 años mientras que la de no-accidente es de 27 a 50 años.

Con estos datos ya encontramos un gran cambio en los resultados que produce el clasificador Random Forest. Conseguimos un 97 % de acierto en el conjunto de entrenamiento tanto para los datos escalados como sin escalar. En el conjunto de testeo tenemos un 61 % de acierto con los datos sin escalar y en el momento que escalamos los datos mejoramos tan solo un 3 % consiguiendo de esta manera un 64 % de acierto.

Conclusiones generales acerca de los resultados

Gracias a las pruebas realizadas, podemos concluir que los clasificadores creados con unos buenos datos predicen bastante bien.

En las dos primeras pruebas se obtienen un acierto mayor del noventa por ciento. Algo lógico ya que hemos creado nosotros los datos a placer y hemos diferenciado mucho las dos clases pero esto nos hace ver que nuestro desarrollo es correcto.

En la segunda prueba, obtenemos un porcentaje mayor. Esto puede ser debido a que al crear el género del individuo se haya generado un conjunto de datos que ayude al clasificador.

En las dos últimas pruebas añadimos dos columnas más y con un mayor número de clases por columna hace que empiece a fallar mas ya que cuando una característica tiene más clases significa que existe más variedad y por tanto que será más difícil encontrar las similitudes para predecir.

Como era de esperar, en la prueba tres y en la prueba cuatro el acierto ya es bastante más bajo rondando el sesenta-setenta por ciento.

Como habíamos comentado anteriormente, escalar los datos ha subido el acierto en todas las pruebas realizadas.

Bajo estas pruebas, hemos llegado a la conclusión que el problema es de los datos con los que hemos trabajado y no nuestro desarrollo de los clasificadores. Seguramente con una muestra de datos más amplia nuestros clasificadores hubieran conseguido mejores resultados.

Capítulo 7

Conclusiones

7.1. Utilidad

Hoy en día, sea cual sea la empresa y sea cual sea el ámbito, si quiere estar actualizada y quiere ser una empresa puntera necesita tener aplicadas técnicas avanzadas informáticas.

La inteligencia artificial, hoy en día está totalmente metida en el mercado y está en pleno auge. Muchas de las mayores empresas del mundo ya están invirtiendo mucho dinero y tiempo en realizar proyectos con técnicas como las usadas en esta aplicación. Por lo que, podríamos decir que este trabajo de fin de grado podría ser un proyecto totalmente real.

Además, en el mundo de los seguros con la cantidad de aseguradoras y de datos sobre los clientes que tienen, consideramos que se pueden hacer grandes predicciones muy potentes y que podría hacer que las empresas ahorraran mucho dinero. Por ello, pensamos que hemos realizado un producto que podría ser muy útil.

Hay que tener en cuenta, que nuestro producto no solo puede ser útil en aseguradoras, sino que la idea base del proyecto, cambiando ciertas propiedades propias de este proyecto, se podría extender en cualquier otro ámbito, teniendo así una idea de producto bastante potente.

Haciendo referencia al ámbito de las aseguradoras, pensamos que hemos facilitado mucho el papel del actuario, y no solo lo hemos facilitado, sino que creemos que de esta manera estaremos ajustando mucho más el riesgo que tiene un cliente ya que tenemos en cuenta factores que no se podrían tener en cuenta sino se estuvieran utilizando estas técnicas.

7.2. Futuras Versiones

En nuestro proyecto, al ser un mundo tan amplio se podrían hacer numerosas versiones futuras que no hemos conseguido implementarlas o bien por motivos técnicos o bien por falta de tiempo.

A continuación, hacemos un desglose de posibles mejoras futuras:

1. Incluir una interfaz más sofisticada y elaborada.
2. Mostrar directamente cual debería ser el precio del seguro del cliente.
3. Dar opciones para que el usuario pueda ver la interfaz en otros idiomas.
4. Obtener datos de los posibles clientes a través de sus redes sociales.
5. Mostrar además del riesgo actual que tiene un cliente, cuál será la evolución de un cliente a lo largo de los años.

6. Mostrar gráficas y un estudio de las predicciones.

Estas ideas son solo algunas de ellas, pero como hemos dicho anteriormente se podrían hacer multitud.

7.3. Valoración del proyecto

Para nosotros, este proyecto ha sido un gran reto el cual pensamos que hemos conseguido superar.

Decimos que ha sido un gran reto, ya que se basaba en una tecnología que nunca habíamos utilizado como es Python. Lo cierto es que el problema no lo tuvimos a la hora de comprender el lenguaje ya que no es difícil y además existe mucha cantidad de documentación sobre él, sino en las técnicas de inteligencia artificial utilizadas las cuales no son sencillas de comprender ni de realizar. Fue aquí donde continuamente nos dábamos contra conceptos complicados y que no éramos capaces de implementar correctamente.

Además, como bien hemos explicado a lo largo de esta memoria, estas técnicas no dan un resultado exacto, ni hay un resultado "correcto". Por tanto, es muy difícil llegar a saber cuál es el mejor resultado y se necesita mucho ensayo-error para saber qué modelo de predicción es el correcto.

Fuera de lo técnico, este proyecto también nos ha ayudado a saber organizar un trabajo costoso y largo lo cual es importante a la hora de afrontar el mundo profesional.

Como autocrítica, pensamos que los clasificadores deberían sacar mejores resultados y así poder tener una aplicación mucho más fiable. También es cierto, que en gran medida nuestros clasificadores no funcionan como nos gustaría ya que el conjunto de datos que tenemos no suficiente y como bien hemos ido explicando a lo largo del documento, los datos son vitales para una buena predicción.

Dejando de lado el párrafo anterior, pensamos que hemos conseguido entender los conceptos de estas tecnologías y hemos llegado a realizar hitos que al principio del mismo nos parecían muy complicados. Esto nos ha dado un gran empujón a la hora de afrontar el mundo laboral y nos ha hecho estar orgullosos del trabajo realizado durante todo el año.

(Versión inglesa en el capítulo 10)

Capítulo 8

Introduction

8.1. Description of the problem

Nowadays, insurance companies base their prices estimation in the actuary's analysis, but this can be very costly. Even though it may look unreal, all the technology and advanced computer techniques that we have access to are not used in this analysis.

To get an idea, the price of an insurance is calculated based on the statistics of two variables: the probability that an unforeseen event occurs, and the average cost resolving it would take. [1]

The person who carries out these calculations is known as "Actuary". As we know, it is not easy to calculate the probability that an unforeseen event occurs, which is the reason why most of the times, insurances are based on a very general client's profile instead of being based on each specific client.

Moreover, different technologies related to Artificial Intelligence are growing nowadays, such as machine learning, neuronal networks, etc. This happens because very positive results are being achieved in the facing of different problems.

Even though lots of these techniques have existed from a long time ago, it is in the current time when they are being applied for real problems solving. The reason why this happens is because of the huge growing of calculation capacity in the last decades, and the data scalability facility that the cloud and big data have created. It is very important to bear in mind that Artificial Intelligence must be completely linked to data, and vice versa.

In view of this lack, our aim is to facilitate the analysis of the actuaries through a computer software based on Artificial Intelligence techniques, and allow them to adapt their work to each specific client.

8.2. Idea of the project

In this project, we have developed a software aimed at supporting insurance companies in achieving a more optimal and secure prediction of the clients' risk, so that the estimated prices for their rates can be more adequate for each client.

To do so, we have used different Artificial Intelligence techniques. The data we have used to make our sorters work correctly were provided by actual insurance companies in *"Excell"* format. These data made our sample consistent enough to get a sorter.

The whole software is implemented in Python3 and developed in Jupyter Notebook environment..

The prediction will be based on different classification models:

1. Neuronal networks
2. Random forest

The insurance companies will have some discretion in deciding how to configure the program. In such way, they will be able to give priority to those characteristics that they consider the most important ones. They will be able to do so in an easy way thanks to our interface, which allows its usage without the necessity of understanding what the program does in order to calculate the predictions.

8.3. Conclusion

As it has been mentioned, it is not easy for the insurance companies to calculate rates for their clients, due to the fact that their clients' risk predictions are based on one person's word instead of using a computer software.

In our opinion, not using the wide variety of tools available is a lack nowadays. This is the reason why we have considered that implementing an innovative tool was a great opportunity for learning more about programming languages that we had never worked with, as well as for learning about an amazing issue, the Artificial Intelligence.

With our product, we will try to help insurance companies by providing a tool that is able to calculate different risk predictions for a client, and, this way, facilitating the actuaries' work who would be able to take these predictions as indicators for establishing rates.

Capítulo 9

Work Planning

In order to achieve our goals, our planning of the project was as follows:

1. Individual study of each member of the group in order to achieve a global understanding of machine learning and the different techniques that have been used in this project.
2. Sharing of information and ideas, and decisions taking regarding the technologies that were going to be used with the other members of the group and the directors and project.
3. Installation and deployment of the tool: Anaconda – Jupyter notebook. (Work Platform).
4. Analysis of the work that had to be done and division of the project among the members of the group.
5. Design of the first drafts with *balsamiq* that allowed an initial vision of the interface.
6. Study of Python, the main language of the program, delving in the different Artificial Intelligence techniques that we had to use for our software.
7. Study of the features that had to be used in the data model to start with the data cleansing.
8. Individual programming with two main sub-aims:
    a. Graphic interface (front-end)
    b. Cleansing and treatment of the data (back-end)
9. Knowledge of a basic neuronal network knowledge with the aim of testing the data treatment.
10. Merging of the two sub-aims to get a first functional software that we could start using for studying the results obtained.
11. Making of tests with the confusion matrix of the neuronal network to evaluate the efficiency of the tool.
12. Improvement of the interface to make it as complete and user-friendly as possible.
13. Deep study of Random forest sorter.
14. Implementation of the other sorters (Random forest).
15. Random forest tests and errors solving.
16. Integration of the Random forest algorithm in the final software.
17. Final software functioning tests and correction of possible mistakes.

It is worth mentioning that this memory was written at the same time the development described above was taking place.

Besides, it is important to highlight that some of these tasks were done by the whole group, but some others were done individually, as it was specified in the planning.

Capítulo 10

Conclusions

10.1. Utility

Nowadays, every company, no matter the field it is specialized on, must use computer techniques in order to be updated.

Artificial Intelligence is completely integrated in today's market, and it keeps growing even more. Many different well-known companies are investing a lot of money and time in carrying out projects with techniques similar to the ones used in this software. Therefore, we could say that this project could perfectly be an actual project.

Besides, in the insurance field, there are lots of companies and data about their clients. This is why very powerful predictions, that would save a lot of money and time to the companies, could be carried out.

It is important to highlight as well that our product could be very useful for other companies different from insurance companies. It could be easily implemented to other fields by taking the base of the project and changing the features.

In relation to the insurance companies, we could state that we have eased the role of the actuary; not only eased it, but getting more accurate predictions of risk for each client, since we have taken into account many features that couldn't have been considered otherwise.

10.2. Future Versions

In our project, lots of future versions could be done, since it is a huge area of study. However, due to technical or time restrictions, we haven't been able to implement them.

Hereunder, future possible improvements are listed:

1. Including a more sophisticated and elaborated interface.
2. Showing directly the rate the client's insurance should have.
3. Give the user the option of using the interface in different languages.
4. Obtaining data about possible clients from their social networks.
5. Showing, apart from the real risk a client has, the evolution this client will suffer along the passing of time.
6. Showing graphics and a study of the predictions.

These are only some ideas, although lots of things could be done in relation to this project.

10.3. Assessment of the Project

For us, this project has been a personal challenge that we think we have been able to deal with.

We consider it a great challenge because it is based on a technology we had never used before, such as Python language. The problem was not understanding the language, since it is not complicated and it is a very documented language, but using Artificial Intelligence techniques that are not easy to understand nor to use.

In addition, as we have explained along this paper, these techniques don't provide an exact result, and "correct" results don't exist either. This is the reason why it is very difficult to know which is the best result and a lot of test-error is needed to know which is the correct prediction model.

Apart from the technical part, through this project we have learned how to organise a big project, which is a very important skill for the labour market.

As a self-criticism, we would say that sorters could obtain better results, and that the software would be much more reliable in such way. It is true that our sorters don't work as good as we would like because we didn't have enough data, which is a basic element to obtain a good prediction.

Finally, we think that we have understood these technologies and we have obtained results that we considered very difficult before starting the project. This is a great support for entering the labour market, since we are proud of the work we have developed along the year.

Capítulo 11

Contribución de cada miembro

- *Aportación Dámaso Sánchez Arenas*

Análisis y aprendizaje de la tecnología

Cuando se comenzó el proyecto, lo primero que hubo que hacer, como se acordó en la primera reunión fue informarse acerca de posibles tecnologías y entornos que podíamos usar.

Esta información principalmente se sacó leyendo en diversos foros de Internet y viendo vídeos de expertos en la materia. La tecnología a usar fue una elección muy acotada ya que Python y R son los lenguajes por excelencia en este tipo de proyectos. Por lo tanto, había que elegir uno de los dos para realizar proyecto. Fue Dámaso el que junto a los directores y la aprobación de Eduardo tomaron la decisión de hacerlo en Python ya que es un lenguaje que tiene mucho interés en el mundo laboral y que está muy demandado hoy en día. Además, se analizó el lenguaje y vimos que podría cumplir con creces todos los requisitos para hacer nuestro proyecto de machine learning.

Una vez elegido el lenguaje y el entorno de trabajo, se comenzó a pensar que bibliotecas debíamos usar. En este caso, Dámaso fue el que tras informarse por Internet decidió hacer la interfaz con la biblioteca de Python *Tkinter* [3]. El motivo principal fue la cantidad de documentación y facilidades que aportaba dicha biblioteca.

Una vez analizado las tecnologías que íbamos a usar, ya estaba todo listo para empezar a desarrollar.

Desarrollo

En el inicio del desarrollo de la aplicación, cuando aún no se había comenzado, con el fin de tener algo funcional lo antes posible, Dámaso comenzó a desarrollar la interfaz gráfica mientras Eduardo comenzaba con la parte back-end. A la par que Dámaso iba desarrollando la interfaz, iba formándose acerca del back-end de la aplicación para después trabajar en paralelo junto a Eduardo.

Una vez que la interfaz ya era lo suficientemente completa como para poder tener algo funcional, Dámaso comenzó a trabajar de manera paralela junto con Eduardo en la parte back-end del proyecto.

Tras un largo periodo trabajando de manera paralela, cuando ya se estaban haciendo las pruebas para ver las predicciones obtenidas por las redes neuronales, Dámaso comenzó a formarse cómo sería el desarrollo sobre el clasificador Random forest.

Una vez que la parte de las redes neuronales estaba apunto de finalizar, y mientras Eduardo daba los retoques finales a la interfaz, Dámaso comenzó a desarrollar la parte del Random forest.

Cuando la parte de las redes neuronales y la interfaz gráfica ya estaban finalizadas, ambos miembros del equipo se centraron en implementar el Random forest y comprobar que los resultados tenían sentido.De esta manera, intentaban poder poner punto final a la parte back-end del proyecto.

Una vez que todo estaba desarrollado, ambos miembros del equipo dedicaron todo su tiempo a realizar pruebas y a corregir fallos.

Gestión de versiones

Las distintas versiones que se iban desarrollando tanto por parte de Eduardo como por parte de Dámaso se iban gestionando usando un repositorio en Github para que continuamente ambos tuviéramos acceso a todo, al igual que los directores.

Documentación

La parte de la documentación se realizó durante todo el proyecto de manera paralela al desarrollo por ambos miembros del equipo. Al final del proyecto, antes de entregar nada, se hizo una revisión de toda la documentación y se solucionaron errores.

La memoria se iba viendo con los directores del proyecto para poder obtener un *feedback* continuo y de esta manera poder ir corrigiendo errores.

- *Aportación Eduardo Rodríguez De Castro Zaloña*

Análisis y aprendizaje de la tecnología

Al inicio del curso, al igual que Dámaso, Eduardo tenía que informarse acerca de las tecnologías y entornos. A pesar de que ambos se informaron paralelamente de ambas cosas, finalmente fue Eduardo el que más se informo acerca de los posibles entornos y el que junto a los directores del proyecto y con la aprobación de Dámaso, decidieron elegir jupyter notebook como entorno de implementación.

Posteriormente, cuando ya se sabía la tecnología y el entorno a utilizar, fue Eduardo el que más esfuerzo inicial puso en informarse sobre el tratamiento del data frame mientras que Dámaso estaba más centrado en la parte de la interfaz. gráfica.

Eduardo había cursado la asignatura de "Minería de Datos" en años anteriores y es por eso que tenía más conocimientos acerca del tema y le costó menos trabajo inicial entender algunos conceptos. Por ello, inicialmente fue Eduardo quien comenzó a desarrollar el back-end de la aplicación.

Desarrollo

Como hemos comentado en el caso de Dámaso anteriormente, Al comenzar el proyecto y con la idea de tener algo funcional cuanto antes, Eduardo comenzó a realizar el back-end de la aplicación gracias a los conocimientos que había adquirido en la asignatura optativa "Minería de datos" mientras Dámaso estaba más centrado en la parte del front-end. Por ello, comenzó a tratar el data frame y a pensar cómo se iban a implementar las redes neuronales. Obviamente, a la vez que iba desarrollando, iba enseñando a Dámaso para que ambos tuvieran conocimiento y pudieran trabajar en paralelo.

Una vez que Dámaso ya obtuvo el conocimiento, durante un buen periodo, ambos miembros del equipo estuvieron trabajando de manera paralela en el tratamiento del data frame y en la implementación de las redes neuronales.

Como bien sabemos, para saber si una predicción se esta haciendo bien, hay que hacer mucho ensayo-error y por eso en esta época del proyecto se necesitaron reuniones constantemente para poder hablar con los directores del proyecto hasta que llegamos a unas redes neuronales que predecían de manera aceptable.

Una vez que ya teníamos una aplicación funcional, aunque no definitiva. Eduardo se centró en corregir fallos de la interfaz e intentar hacerla lo más amena y práctica posible.

Por último, cuando tanto las partes de las pruebas con las redes neuronales como las de la interfaz gráfica estaban realizadas, Eduardo junto con Dámaso desarrollaron y finalizaron la parte del clasificador Random forest.

Una vez que todo estaba desarrollado, ambos miembros del equipo dedicaron todo su tiempo a realizar pruebas y a corregir los posibles fallos.

Gestión de versiones

Las distintas versiones que se iban desarrollando tanto por parte de Eduardo como por parte de Dámaso se iban gestionando usando un repositorio en Github para que continuamente ambos tuviéramos acceso a todo, al igual que los directores.

Documentación

La parte de la documentación se realizó durante todo el proyecto de manera paralela al desarrollo por ambos miembros del equipo. Al final del proyecto, antes de entregar nada, se hizo una revisión de toda la documentación y se solucionaron errores.

La memoria se iba viendo con los directores del proyecto para poder obtener un *feedback* continuo y de esta manera poder ir corrigiendo errores.

Bibliografía

[1] conocimiento acerca del mundo de los seguros, Enlace: https://www.estamos-seguros.es/como-se-calcula-el-precio-del-seguro/ 5, 49

[2] página para realizar los mock-ups iniciales, Enlace: https://balsamiq.com/ 12

[3] biblioteca tkinter , Enlace: https://docs.python.org/2/library/tkinter.html 54

[4] Biblioteca Panda, Enlace: https://pandas.pydata.org/pandas-docs/stable/ 31

[5] biblioteca numpy, Enlace: https://docs.scipy.org/doc/

[6] videotutoriales acerca de la implementación de la interfaz con la biblioteca tkinter, Enlace: https://www.youtube.com/watch?v=G2FCfQJ-9ig&list=PLU8oAlHdN5BlvPxziopYZRd55pdqFwkeS

[7] evolución histórica de las redes neuronales, Enlace: http://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/ 27

[8] concepto acerca de las redes neuronales, Enlace: https://es.wikipedia.org/wiki/Red_neuronal_artificial 28

[9] información acerca de las redes neuronales, Enlace: https://www.xataka.com/robotica-e-ia/las-redes-neuronales-que-son-y-por-que-estan-volviendo 28

[10] implementación de las redes neuronales, Enlace: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html 36

[11] evolución histórica del clasificador random forest, Enlace: https://es.wikipedia.org/wiki/Random_forest 29

[12] concepto acerca del random forest, Enlace: https://bookdown.org/content/2031/ensambladores-random-forest-parte-i.html

[13] información acerca del random forest, Enlace: http://games.cmm.uchile.cl/media/uploads/posts/RF_presentation.pdf

[14] implementación del clasificador random forest, Enlace: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[15] implementación de una matriz de confusión, Enlace: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html 37

[16] implementación del escalado de datos, Enlace: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html 34

[17] implementación del entrenamiento de un clasificador, Enlace: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html 37

[18] consejos acerca del machine learning, Enlace: https://machinelearningmastery.com/framework-for-better-deep-learning/

[19] Como implementar de manera efectiva técnicas de machine learning, Enlace: https://machinelearningmastery.com/

[20] solución de dudas particulares en toda la aplicación, Enlace: https://es.stackoverflow.com